Master's Thesis

AN EMPIRICAL ANALYSIS OF EVOLUTIONARY PARTICLE SWARM OPTIMIZATION

JEROEN VAN HOEK

Utrecht University

Department of Information and Computing Sciences
Faculty of Science
November 2017

ing. Jeroen van Hoek: *An Empirical Analysis of Evolutionary Particle Swarm Optimization*, © November 2017

SUPERVISOR:
dr. G.A.W. Vreeswijk

LOCATION:
Utrecht

ABSTRACT

Evolutionary Particle Swarm Optimization (EPSO) is a combination of Classic Particle Swarm Optimization and Evolutionary Algorithms. In this thesis we recreate the original EPSO algorithm and analyze its performance. Furthermore we extend the algorithm with uniform recombination, another idea from Evolutionary Algorithms. Finally we analyze why EPSO performs so well, and whether our extension fixes the mistakes made by EPSO or not.

The motion of a flock of birds is one of nature's delights. Flocks and related synchronized group behaviors such as schools of fish or herds of land animals are both beautiful to watch and intriguing to contemplate.

— Craig W. Reynolds [15]

ACKNOWLEDGMENTS

First and foremost I would like to thank my supervisor, dr. Gerard Vreeswijk for his support and feedback during the writing of this thesis. Secondly I would like to thank Jonathan Reid for his input on my writing style, help with proof-reading, and his opinion on various decisions made throughout the writing process. Lastly I would like to thank everyone else who supported me in various other respects while I was working on this thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The flight of birds has fascinated humans since time immemorial, and will most likely still fascinate humans for a long time to come. While some dreamt of taking to the skies themselves, others preferred to stay grounded. With the dawn of computer simulations, the flight of birds could be recreated digitally. However, not all research went into the actual flight mechanics of the birds. Research on the flocking behavior of birds [7] was an inspiration for particle swarm optimization (PSO) [8]. In this thesis we will use the term 'Classic PSO' when referring to this PSO version by Kennedy.

An evolutionary algorithm (EA) usually consists of a population of solutions that apply techniques borrowed from Darwinian evolutionary theories to solve complex problems [5]. Such techniques can include mutation, reproduction, selection and more. Reproduction is often categorized as either asexual or sexual reproduction, generating new solutions from a single or multiple parents respectively. Selection is based on the principles of natural selection, often a tournament-style selection is applied to choose the best solution that gets to survive to the next generation.

Evolutionary particle swarm optimization (EPSO) is a combination of PSO and EAs that applies mutation and selection from EAs in addition to the Classic PSO movement [10–12]. With these additions, a significant gain in performance was observed. While the results gathered with this algorithm seem promising, we fear the authors attributed the performance to the wrong aspects of their algorithm. While mutation should improve the performance of Classic PSO, we believe that other deviations of the Classic PSO algorithm are responsible for their performance numbers.

Our goal is to examine the performance of EPSO through recreation and expansion of the algorithm. We aim to extend EPSO with recombination from Evolutionary Algorithms, and to test the impact of various EPSO aspects on the performance of EPSO. Finally we are going to examine whether our own expanded algorithm performs better than EPSO in various scenarios.

2 EVOLUTIONARY PARTICLE SWARM OPTIMIZATION

Mutation is the motor of evolution. It is the source for the variation of the genome. Therefore, never underestimate the power of mutation: An EA without recombination can work, whereas an EA without mutations will get stuck.

— H. G. Beyer [2]

EPSO is a combination of Classic PSO [8] and Evolutionary Algorithms [5]. EPSO combines these two by expanding Classic PSO with the mutation aspect from Evolutionary Algorithms. To better understand EPSO and its inner workings this chapter explains the different components of EPSO, showing where mutation is applied and how it works exactly in this context. Furthermore, it explains the difference between EPSO and Classic PSO and how we recreated EPSO. Some variable names have been changed from the original papers to be consistent with the rest of this thesis.

In EPSO each swarm contains:

- $k \in \mathbb{Z}^+$ number of particles.
- g_{best}, $\vec{g}_{best}$: the global best found value $\in \mathbb{R}$ and the corresponding position vector with values $\in \mathbb{R}$.

Each particle consists of:

- p_{best}, $\vec{p}_{best}$: personal best encountered value $\in \mathbb{R}$ and the corresponding position vector with values $\in \mathbb{R}$.
- $\vec{x}$: position vector with values $\in \mathbb{R}$ in the search space.
- $\vec{v}$: velocity vector (inertia) with values $\in \mathbb{R}$.
- $w^k_{inertia}, w^k_{memory}, w^k_{cooperation} \in \mathbb{R}$: strategic parameters (weights) for inertia, memory and cooperation respectively.

At each evaluation in the algorithm the particles update their p_{best} if they have found a better position. This value is then checked against the g_{best} of the swarm, and that is updated if the value improves. If g_{best} is updated every particle will learn this information in the next iteration.

In every iteration the EPSO algorithm executes the following 5 steps:

1. Replication: each particle in the current swarm is replicated r times, creating r sets of identical particles. In the original EPSO algorithm $r = 1$ is used.

2. Mutation: the weights of each replicated particle are mutated with:

$$^{*}w_j^k = w_j^k + \tau N(0,1) \tag{2.1}$$

In the beginning of the algorithm the strategic parameters of each particle are randomly set between 0 and 1. In each iteration the mutation formula changes the inertia, memory and coordination weight parameters j of each particle k. $^{*}w_j^k$ is the new strategic parameter for particle k. Where τ is a Learning Dispersion parameter[1] and $N(0,1)$ is a random number from a Gaussian distribution with mean 0 and variance 1.

3. Reproduction (movement): for each particle k, offspring is generated with movement similar to the Classic PSO formula:

$$\begin{aligned} ^{*}\vec{v}^k = {} & w_{inertia}^k \, \vec{v}^k \\ & + w_{memory}^k \, (\vec{p}_{best}^k - \vec{x}^k) \\ & + w_{cooperation}^k \, (\vec{g}_{best}'' - \vec{x}^k) \end{aligned} \tag{2.2}$$

$$^{*}\vec{x}^k = \vec{x}^k +^{*} \vec{v}^k \tag{2.3}$$

$^{*}\vec{v}^k$ is the new velocity vector for particle k, and $^{*}\vec{x}^k$ is the new position vector of the particle k. $\vec{p}_{best}^k$ is the location vector of the personal best of the particle k. This movement is applied to both the original and the mutated particles.

Instead of being attracted to the exact best-so-far point, the particles are attracted to a "foggy best-so-far region" which is another deviation from Classic PSO. This is done by introducing random noise to the best-so-far point with:

$$\vec{g}_{best}'' = \vec{g}_{best} + \tau' N(0,1) \tag{2.4}$$

$\vec{g}_{best}''$ is the mutated best-so-far point, also known as the best-so-far region. τ' is a noise dispersion parameter[2], and $N(0,1)$ is another random number from a normalized Gaussian distribution with mean 0 and variance 1.

4. Evaluation: the mutated and original particles are evaluated according to their position vector in the search space and the test function used.

5. Selection: among the mutated and original particles, a stochastic tournament selection is played to select the particle that will survive to the next generation. However since $r = 1$ is used the tournament is not stochastic, there are only two particles in each tournament thus they must compete with each other.

1 Discussed further in Section 2.2.
2 Discussed further in Section 2.2.

The two main differences between EPSO and Classic PSO lie in step 2 and 3. In the mutation step changing the learning parameters adds a level of randomness to the movement the particles apply. By mutating one and keeping the other unchanged there is more exploration of the target position. Mutating the strategic parameters has proven to be a very effective exploration method [3]. Since all strategic parameters can mutate, every vector in the movement calculation is affected.

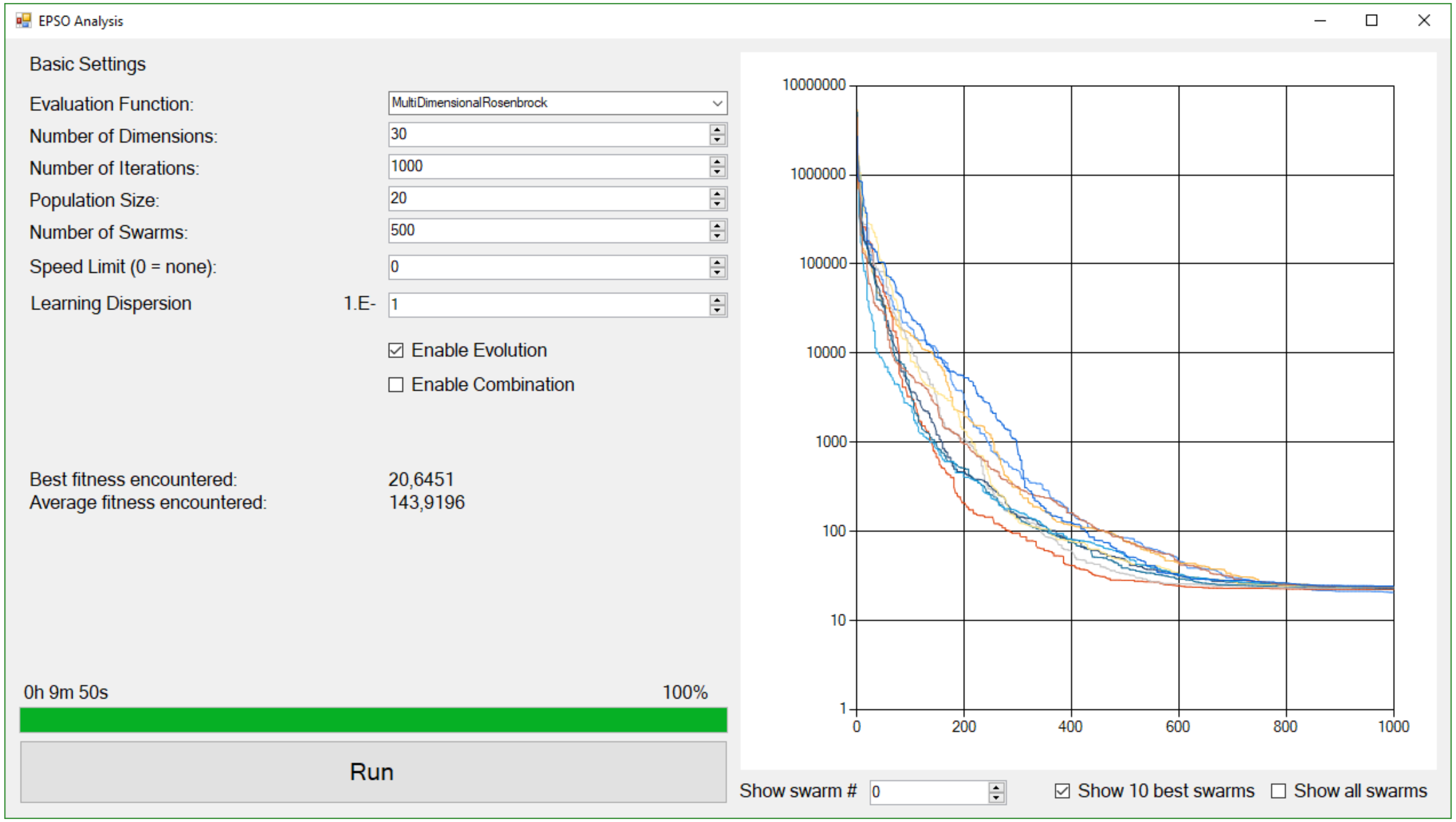
Another deviation from Classic PSO is observed in the movement step, particularly the "foggy best-so-far region". This foggy attraction point for the best found solution is essentially another introduction of randomness to the algorithm. In theory this works much like the mutation of the strategic parameters, however since it's done for both the original and the mutated particles it ensures even more exploration of the search space.

2.1 RECREATING EPSO

In order to recreate EPSO we first built the basic PSO algorithm according to the original paper by Kennedy & Eberhart [8]. After a basic proof of concept was made that performed similarly to the PSO algorithm described in the EPSO paper [12], it was extended to incorporate the EPSO adjustments. During the recreation of EPSO the performance of the algorithm was monitored to stay in line with the original EPSO results. The performance was monitored on the Binary Schaffer, Rosenbrock and Sphere optimization functions as described in Appendix A similarly to the EPSO papers.

In order to monitor performance and set up tests quickly, we created a graphical user interface (GUI) that could render graphs of swarm convergence quickly. This allowed us to visualize results on the influence of different settings on EPSO without exporting the data to analytics software first. Figure 2.1 shows a screenshot of what our GUI looks like after a test run[3]. The right hand side shows a graph of the 10 best swarms of the collection, while the left side shows the settings and test progress. Note that the setting of Learning Dispersion has a $1.e-$ prefix, this was added for improved readability, but needs to be kept in mind while inputting settings for a test run. The GUI also allows for single-swarm display, by selecting the index of the swarm below the graph. This is useful to identify oddities in the results and to investigate those further. On completion of a test run our program stores all the data points in a .csv file that can then be analyzed with programs such as MATLAB [9] and R [19].

3 Specifically the test on the Rosenbrock function discussed in Section 2.3.



Basic Settings

Evaluation Function:	MultiDimensionalRosenbrock
Number of Dimensions:	30
Number of Iterations:	1000
Population Size:	20
Number of Swarms:	500
Speed Limit (0 = none):	0
Learning Dispersion 1.E-	1

☑ Enable Evolution

☐ Enable Combination

Best fitness encountered:	20,6451
Average fitness encountered:	143,9196

0h 9m 50s 100%

Figure 2.1: A screenshot of our testing user interface, the graph shows the best fitness encountered per iteration on a logarithmic scale.

2.2 MISSING PARAMETERS

During the recreation of EPSO we noticed that not every part of the algorithm is as well defined as it appears. While the description is sufficient for reading, it is crucial to get every aspect exactly right in the process of recreation. In Chapter 2 the "foggy best-so-far region" used in EPSO is explained. The mutation formula uses a Learning Dispersion parameter τ that is never fully defined. Another parameter like this is the 'Noise Dispersion' used in step 3 of the EPSO algorithm. This parameter was described as *"τ' is a noise dispersion parameter, usually small"* [12]. These parameters were both unclear in their description, and had to be explored to estimate their actual values.

In order to estimate Learning Dispersion the Sphere and Rosenbrock functions was used to experiment with different values of τ. From the way Learning Dispersion is used and described it was estimated that τ was somewhere in the range of $[0, 1]$. Early testing showed that values above 0.5 provided significantly worse results than lower values, thus testing was focused on values under 0.5. These tests were concluded on a fixed 1000 iterations per parameter, with 20 particles per swarm and 500 swarms per test. For each test, the results of the best swarm is used.

A similar approach was used for the Noise Dispersion parameter. However, it was noted that the Noise Dispersion has a much higher tolerance for variation, and the data on this was not as interesting. The conclusion for Noise Dispersion is that the results get progressively worse the lower the value of τ', and any value in the range of 0.05 to 0.75 provides decent results on both the Rosenbrock and Sphere functions. For all following experiments, τ' was set to 0.75.

2.3 ROSENBROCK FUNCTION ON LEARNING DISPERSION

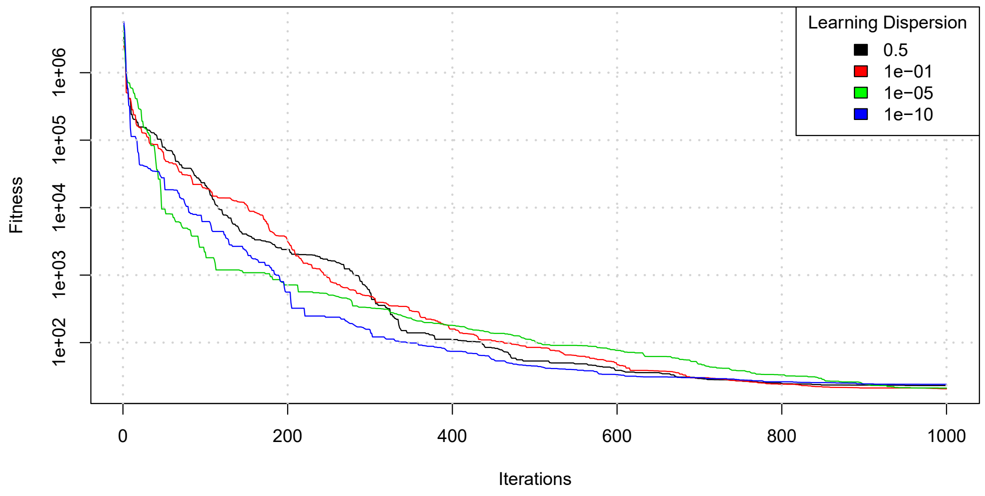


Figure 2.2: The convergence of minima with different values for Learning Dispersion on the Rosenbrock function. Note that the y-axis uses a logarithmic scale.

The minima found by the swarms on the Rosenbrock function did not differ much as seen in Figure 2.2. However the difference in performance becomes more obvious when looking at the means of the different functions. In Table 2.1 it is clearly visible that a Learning Dispersion of $1.e-01$ shows the best performance, and should be used on all following experiments on the Rosenbrock function.

τ	Average	Minimum
0.5	1 051.724	23.093
$1.e-01$	**143.920**	**20.645**
$1.e-05$	253.791	21.185
$1.e-10$	261.507	24.080
$1.e-15$	244.083	25.889

Table 2.1: The means and minima found with the different Learning Dispersion values on the Rosenbrock function. Bold text is used to denote the best values.

2.4 SPHERE FUNCTION ON LEARNING DISPERSION

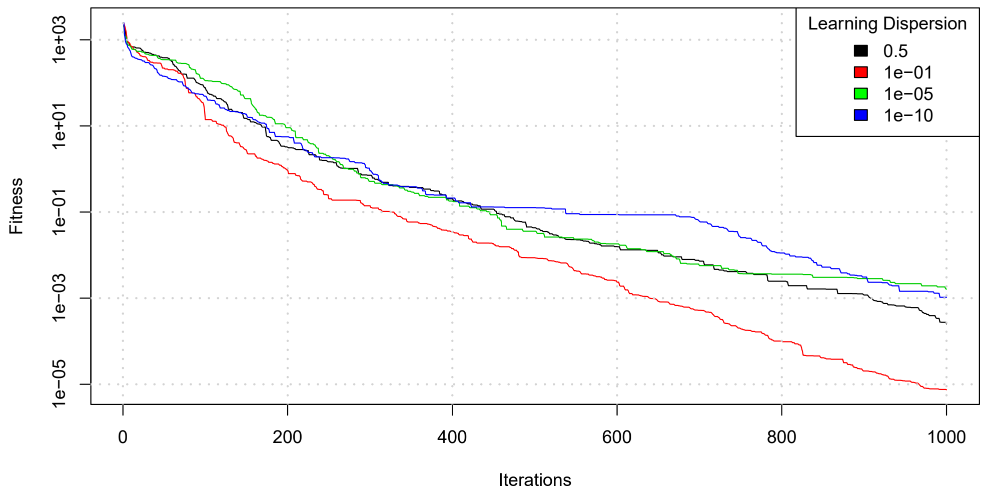


Figure 2.3: The convergence of minima with different values for Learning Dispersion on the Sphere function. Note that the y-axis uses a logarithmic scale.

On the Sphere function the difference in minima is much larger, with $1.e-01$ showing a clear advantage over the other values in Figure 2.3. In Table 2.2 the superiority of $1.e-01$ is visible as well, with the lowest minimum found and a much lower mean value. Therefore $1.e-01$ should be used as the parameter in the Sphere function as well.

τ	Average	Minimum
0.5	2.871	$2.523e-04$
$1.e-01$	**0.194**	$\mathbf{7.467e-06}$
$1.e-05$	0.677	$1.622e-03$
$1.e-10$	1.159	$1.046e-03$
$1.e-15$	0.970	$1.785e-03$

Table 2.2: The means and minima found with the different Learning Dispersion values on the Sphere function. Bold text is used to denote the best values.

2.5 LEARNING DISPERSION CONCLUSION

The average results on both the Rosenbrock and Sphere functions are greatly improved when τ is set to $1.e-01$, compared to all other tested values. The minima on the Rosenbrock function do not show such a large difference. However, the algorithm shows a more robust performance on average. The algorithm shows improvement on the sphere function in both average and minimum results, thus we concluded that the Learning Dispersion parameter should be set to $1.e-01$ for future experiments. On other problems the parameter could possibly be tweaked to improve problem-specific performance. However, one of the main arguments for EPSO is that parameter tweaking should not be necessary and therefore we concluded that in EPSO Learning Dispersion should be set to $1.e-01$.

3 EXPANDING EPSO

The core concept from Evolutionary Computation that EPSO incorporates is particle 'mutation' from genetic algorithms. Mutation is common in genetic algorithms, however there are other methods available for creating more exploration of the search space. Some genetic algorithms use crossover or 'combination' to generate new solutions by combining existing ones. Some notable crossovers include 1-point, 2-point and uniform crossover. 1-point crossover takes 1 to k variables from parent p_1 and k to n variables from parent p_2. with k being the crossover point either fixed or randomized, and n number of variables in the solution vector. 2-point crossover works similarly, adding in a second crossover point l giving the child solution 1 to k variables from p_1, k to l from p_2 and l to n from p_1 again [5].

Uniform crossover disposes of the crossover points for an even more randomized approach. The concept of 2 parents stays the same but it adds a variable P_1, the chance a specific variable is taken from p_1. With $P_1 = 0.5$ it essentially becomes a coin toss to determine which parent gives which variable to the offspring. This uniform recombination of two parents disrupts the search space more than 1-point and 2-point crossover [18]. This disruption results in more exploration of the search space. An example of uniform crossover is shown in Table 3.1. *"With small populations, more disruptive crossover operators such as uniform or n-point ($n > 2$) may yield better results because they help overcome the limited information capacity of smaller populations and the tendency for more homogeneity."* [17]. With this in mind we selected uniform crossover with P_1 set to 0.5 in our testing.

Index	0	1	2	3	4	5	6	7	8	9
Parent A	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Parent B	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
$P_1 = 0.5$	F	F	F	T	F	T	T	T	F	T
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Child	🟥	🟥	🟥	🟦	🟥	🟦	🟦	🟦	🟥	🟦

Table 3.1: Uniform crossover

3.1 APPLYING CROSSOVER

To implement a crossover step into EPSO the following steps were introduced between step 2 (mutation) and step 3 (movement) of EPSO to generate a child solution:

1. Selection: select p_1 parent by index, select p_2 at random from the population.
2. Parameters: Strategic parameters (weights) are copied from p_1.
3. Crossover: Child's position in the search space is determined by uniform crossover on the parent solutions.
4. p_{best}: The combined solution is evaluated to generate a preliminary personal best value.

Selecting the first parent by index ensures that each particle will generate at least one child. Combined with a random second parent more combinations are explored than with two fixed parents. Because we know that each particle will get at least one child, copying the strategic parameters can be done without risking the loss of a useful parameter set. Uniform crossover is applied with $P_1 = 0.5$. The child is evaluated once for two reasons: to generate a p_{best} it can use in future generations for movement and secondly update the g_{best} if required. After generating a set of children, they are included in the rest of the EPSO steps just as the mutated particles are. The only extra change to EPSO is step 6 (selection): Instead of tournament selection with $n = 2$ over the original and mutated particles, tournament selection with $n = 3$ is used over the original, mutated and combined particles. So for each parent particle one mutated particle and one child particle are evaluated and the best of the 3 stays in the population for the next generation.

3.2 LIMITING PARTICLE SPEED

It was like watching spacecraft explore the Milky Way Galaxy in order to find a target known to be in the Solar System.

— R. C. Eberhart & Y. Shi [4]

Because of the inertia used in Classic PSO, particles can gain ever increasing speed at which they traverse the solution. This speed can cause the particles to oscillate around certain points if no improvements to g_{best} and p_{best} are found. This oscillation can cause the particles to get stuck in their search prematurely. One solution to prevent oscillation from happening too early is to limit the distance a particle can move over a certain parameter per iteration [14]. One issue with such a speedlimit is that a speedlimit is a problem specific parameter and it would need tweaking for every test function. However, as Everhart and Shi discussed [4] limiting the speed to the largest dimension of distance across the solution space would make sense. Allowing a particle to explore multiples of the search space in one step has no added benefit, and might hamper performance. Therefore, when applying a speedlimit, we chose to limit the speed to the width of the domain of each function. We used this speedlimit PSO to compare with the other optimization algorithms because it showed greatly improved performance in preliminary testing over Classic PSO without increasing the computational cost.

4 TESTING EPSO EXPANSIONS

Since both the standard EPSO and the Combination algorithm are in essence EPSO algorithms, EPSO is referred to as 'Evolution' in the test results. To be able to compare the different algorithms to each other, experiments were conducted with a fixed number of evaluations for each test function. Basic PSO and PSO with a speedlimit serve as the baseline number of evaluations, Evolution requires twice as many evaluations and Combination requires three times the number of evaluations. This is due to the mutation step in Evolution and Combination and the evaluation of offspring in Combination. To select the cutoff point for number of evaluations we looked at the convergence speed of the algorithms, to show convergence but not let the algorithms compute forever. The algorithms were limited to 1500 evaluations on the Schaffer F2 function and 3000 evaluations on the Schaffer F6 function. For the Rosenbrock and Sphere functions each algorithm was limited to 150 000 evaluations. Each test was conducted with 500 swarms per test function, and 20 particles per swarm.

4.1 OPTIMIZATION ON THE SCHAFFER F2 FUNCTION

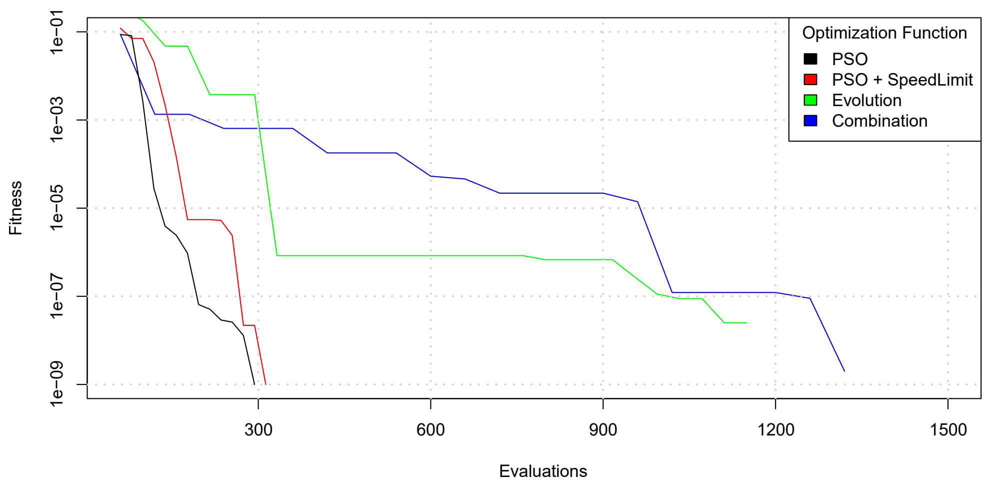


Figure 4.1: The convergence of the different methods on the Schaffer F2 function. Note that the y-axis uses a logarithmic scale.

The results presented in Figure 4.1 and Table 4.1 show that in 1500 evaluations all 4 methods manage to find the global optimum. The main difference can be seen in the mean result, where Combination show superiority over all other algorithms. Evolution manages to pull ahead of the traditional 2 methods as well, yet performs worse than Combination.

Because every algorithm finds the global optimum the graphs plotted in Figure 4.1 are the swarms that reached the global optimum in the least amount of iterations. The plot lines terminate where the swarms have reached 0. While the traditional PSO algorithms find the global optimum faster than Evolution and Combination they still perform worse on average.

Method	Average	Minimum
PSO	$4.510e-03$	0
SpeedLimit	$1.541e-03$	0
Evolution	$2.697e-05$	0
Combination	$\mathbf{1.949e-08}$	0

Table 4.1: The means and minima found by the different optimization methods on the Schaffer F2 function. Bold text is used to denote the best values.

4.2 OPTIMIZATION ON THE SCHAFFER F6 FUNCTION

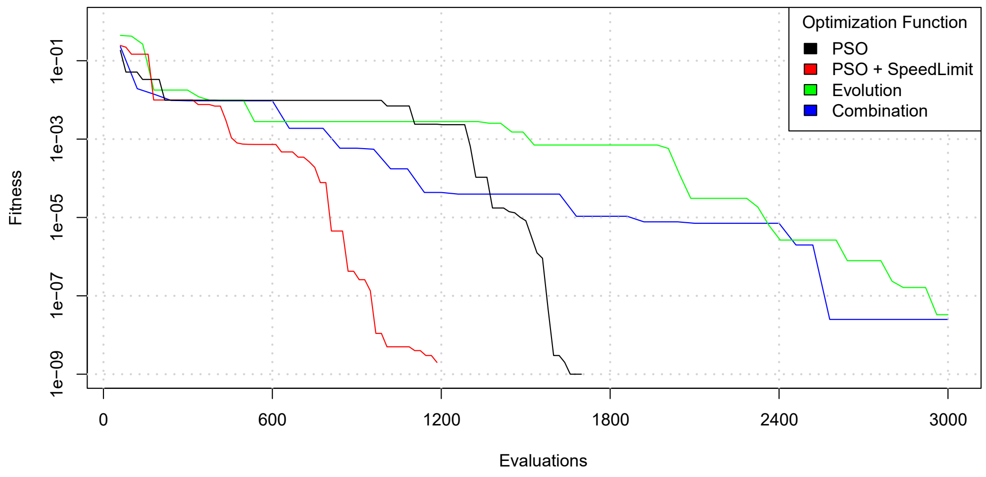


Figure 4.2: The convergence of the different methods on the Schaffer F6 function. Note that the y-axis uses a logarithmic scale.

The results presented by Figure 4.2 and Table 4.2 show that Evolution finds the best average result. However, the standard PSO and SpeedLimit algorithms find the global optimum, while the more complex Evolution and Combination do not. Combination and Evolution algorithms show similar results, and while they do not manage to find the global optimum they still show robustness in their average results.

Method	Average	Minimum
PSO	$1.021e-02$	0
SpeedLimit	$9.113e-03$	0
Evolution	**$8.203e-03$**	$3.3e-08$
Combination	$1.142e-02$	$2.5e-08$

Table 4.2: The means and minima found by the different optimization methods on the Schaffer F6 function. Bold text is used to denote the best values.

4.3 OPTIMIZATION ON THE ROSENBROCK FUNCTION

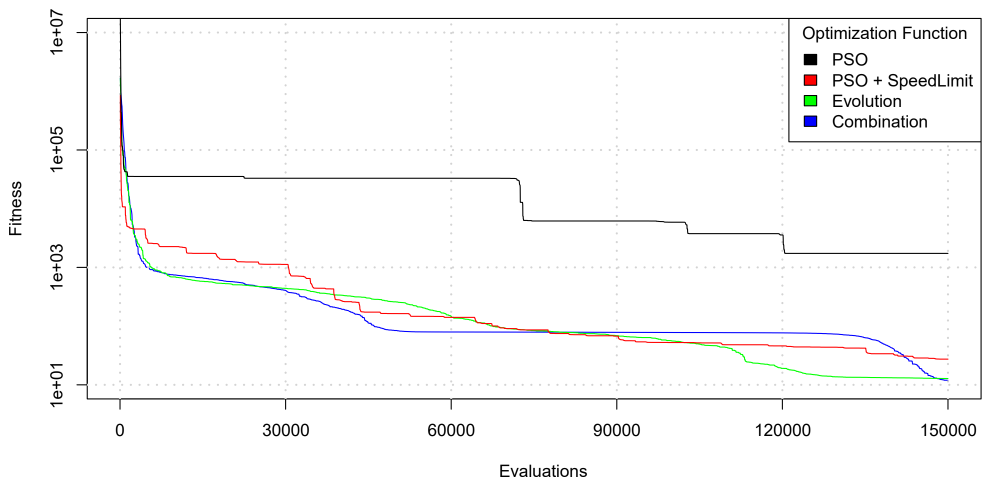


Figure 4.3: The convergence of the different methods on the Rosenbrock function. Note that the y-axis uses a logarithmic scale.

The Rosenbrock function is where the Classic PSO algorithm really starts to struggle. In Figure 4.3 and Table 4.3 the inferiority of Classic PSO is demonstrated. SpeedLimit, Evolution and Combination are much closer together in their results. The average result of PSO is particularly poor, showing that the minimum encountered is more result of chance than performance. The results of the Evolution and Combination methods do not differ much from each other in Table 4.3. In Figure 4.3 the similarity of the convergence of the Evolution and Combination algorithms is visible as well.

Method	Average	Minimum
PSO	97 604.412	1 731.415
SpeedLimit	380.969	27.381
Evolution	47.276	12.799
Combination	**38.265**	**11.910**

Table 4.3: The means and minima found by the different optimization methods on the Rosenbrock function. Bold text is used to denote the best values.

4.4 OPTIMIZATION ON THE SPHERE FUNCTION

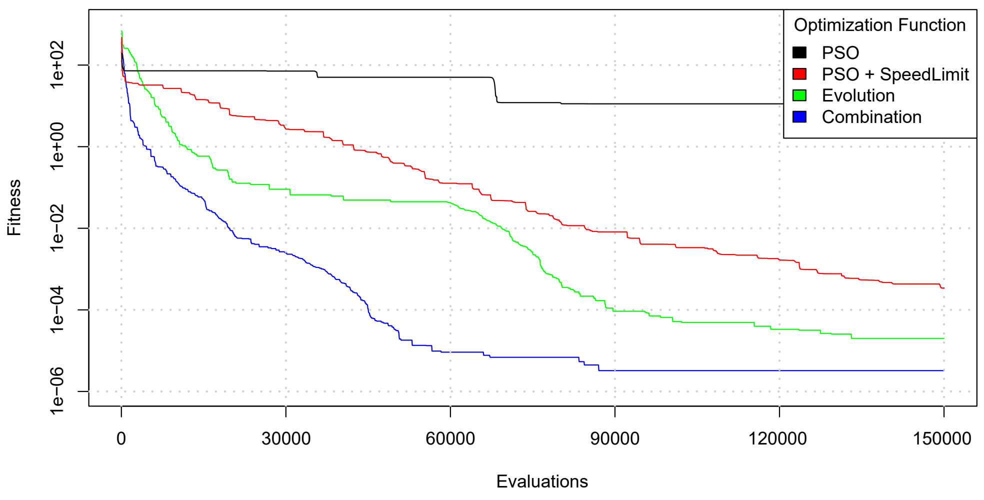


Figure 4.4: The convergence of the different methods on the Sphere function. Note that the y-axis uses a logarithmic scale. The PSO graph line continues behind the legend, but does not improve further.

On the Sphere function the largest difference between the 4 algorithms is visible. In Figure 4.4 and Table 4.4 the superiority of Evolution and Combination is demonstrated. Again PSO suffers and the average result is much higher than the other algorithms, demonstrating it is much less robust than the more complex methods. Evolution shows the best average result, however Combination is not far behind and shows a slightly improved minimum result. In Figure 4.3 the plot of Combination converges much faster than Evolution does, and with a lower cutoff for maximum evaluations[1] the Combination algorithm would have significantly better results, however this comparison is moot with a fixed number of evaluations.

1 60 000 iterations specifically.

Method	Average	Minimum
PSO	88.938	5.026
SpeedLimit	$7.845e-01$	$3.422e-04$
Evolution	$\mathbf{9.561e-02}$	$1.999e-05$
Combination	$1.019e-01$	$\mathbf{3.243e-06}$

Table 4.4: The means and minima found by the different optimization methods on the Sphere function. Bold text is used to denote the best values.

5 ANALYZING FOGGY APPROACHES

During the development and testing of EPSO we noticed that the mutation of the best-so-far point into a "foggy best-so-far region" impacted performance of EPSO greatly. This quickly led to the hypothesis that EPSO might rely on this "foggy best-so-far" *too* much, and might be less robust without it. Thus testing was devised to measure the impact of this 'fog' on the algorithm results. It has been shown that the PSO + Speedlimit algorithm performs equal or better on all test functions in Chapter 3, therefore the results of the Classic PSO have been omitted. The remaining 3 algorithms are tested on an equal amount of evaluations, with 20 particles per swarm and 500 swarms per test function.

5.1 SCHAFFER F2 WITHOUT FOG

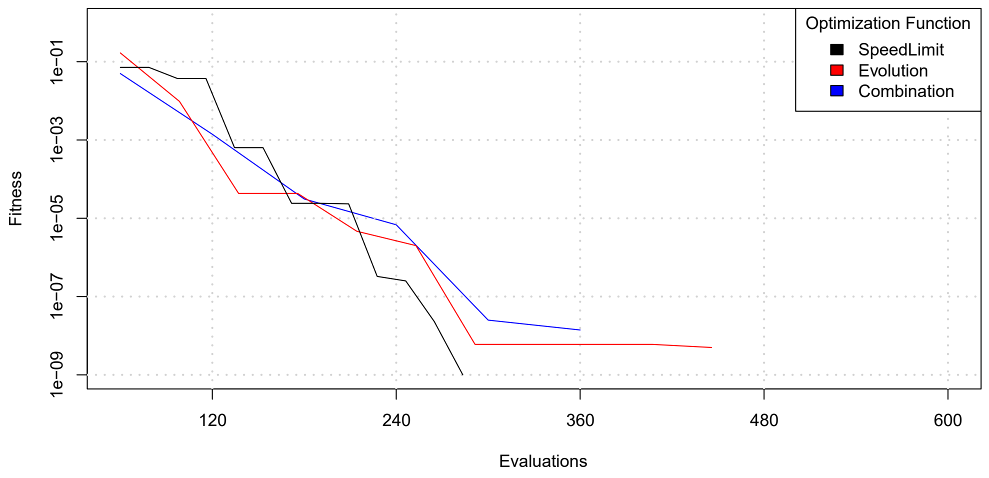


Figure 5.1: The convergence of different methods on the Schaffer F2 function with the fog disabled. Note that the y-axis uses a logarithmic scale.

In the Schaffer F2 function the 3 algorithms do not show much difference in average, minimum or convergence behaviour in Table 5.1 and Figure 5.1. This shows that without the "foggy-best-so-far region" the addition of mutation and combination does not improve the algorithm noticeably. In this test function the Evolution and Combination algorithms show that without the fog they are not much different from the Speedlimit algorithm, or from each other.

Method	Average	Minimum
SpeedLimit	$1.009e-03$	0
Evolution	$\mathbf{5.949e-04}$	0
Combination	$2.678e-03$	0

Table 5.1: The means and minima found by the different optimization methods on the Schaffer F2 function with the fog disabled. Bold text is used to denote the best values.

5.2 SCHAFFER F6 WITHOUT FOG

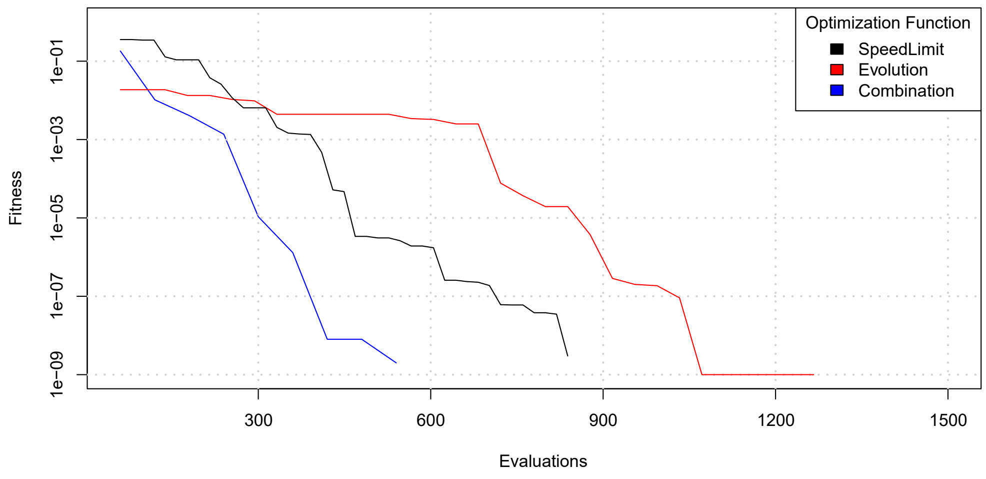


Figure 5.2: The convergence of different methods on the Schaffer F6 function with the fog disabled. Note that the y-axis uses a logarithmic scale.

Again the 3 compared methods do not show much difference in average result in Table 5.2. Only in Figure 5.2 some difference in convergence is visible. This suggests that the algorithms are very similar without the "foggy-best-so-far region" on this function as well.

Method	Average	Minimum
SpeedLimit	$9.265e-03$	0
Evolution	$\mathbf{1.016e-02}$	0
Combination	$1.543e-02$	0

Table 5.2: The means and minima found by the different optimization methods on the Schaffer F6 function with the fog disabled. Bold text is used to denote the best values.

5.3 ROSENBROCK WITHOUT FOG

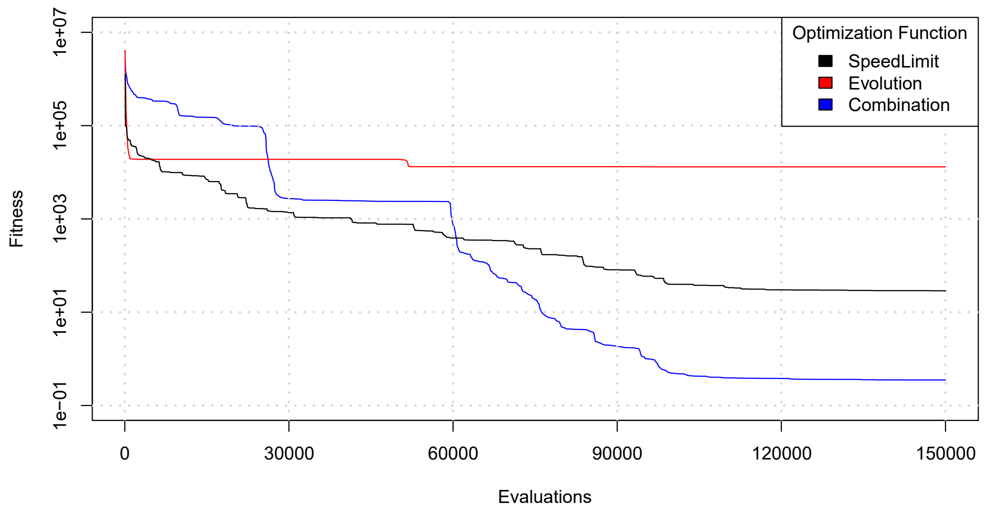


Figure 5.3: The convergence of different methods on the Rosenbrock function with the fog disabled. Note that the y-axis uses a logarithmic scale.

On the Rosenbrock function the differences between the 3 algorithms become more pronounced in Figure 5.3. The SpeedLimit function still performs as solidly as it did before, while the Evolution algorithm struggles to reach an optimum. Combination manages to show a superior best result in Table 5.3 even compared to previous testing, however the average result is still worse than with the "foggy-best-so-far region" implemented. This shows that both the Evolution and Combination algorithms rely on fog to some extent and that they are much less robust without it.

Method	Average	Minimum
SpeedLimit	**266.374**	28.817
Evolution	146 280.0	13 085.08
Combination	22 016.69	**0.353**

Table 5.3: The means and minima found by the different optimization methods on the Rosenbrock function with the fog disabled. Bold text is used to denote the best values.

5.4 SPHERE WITHOUT FOG

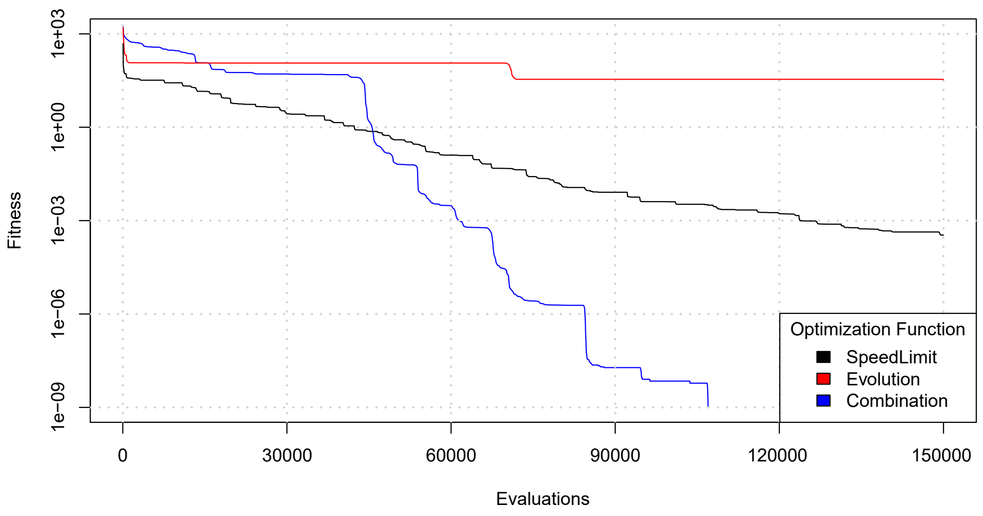


Figure 5.4: The convergence of different methods on the Sphere function with the fog disabled. Note that the y-axis uses a logarithmic scale.

Unsurprisingly the SpeedLimit algorithm is still very robust in this testing, showing the best average result in Table 5.4. Combination shows clear superiority in finding the optimum, with an average result close to the minimum found by Evolution. The Evolution method without the "foggy-best-so-far region" is much less robust than the original, visible in Figure 5.4, hinting that the algorithm was relying too much on the randomness introduced there. While the Combination algorithm also shows worse results in this comparison, it is still capable of finding the global optimum.

Method	Average	Minimum
SpeedLimit	**0.784**	$3.422e-04$
Evolution	347 711	34 396
Combination	34 379	**0**

Table 5.4: The means and minima found by the different optimization methods on the Sphere function with the fog disabled. Bold text is used to denote the best values.

5.5 ANALYSIS CONCLUSION

From the tables and graphs in the previous sections we can conclude that without the "foggy-best-so-far region" Evolution performs significantly worse overall. The Evolution algorithm still performs better on average than Classic PSO on the Rosenbrock function, yet it performs worse than Classic PSO on the Sphere function. This shows that the mutation of strategic parameters is not a 'catch-all' solution for every problem. While the mutation works well in combination with the randomness from a "foggy-best-so-far region", it does not improve results that much without the fog. Furthermore, it is also shown that the Combination algorithm suffers a heavy performance decrease as well. However, it shows more robustness than the Evolution algorithm does. This indicates that the Combination approach might generally be a better one than the Mutation approach from Evolution. Tweaking parameters such as the Learning Dispersion might have influence on these results, however one of the key selling points of EPSO is the lack of need for parameter tweaking. This could be investigated, but it is left for future work.

6 CONCLUSION & DISCUSSION

In this thesis we have successfully extended the EPSO algorithm as described in [12] and [11] with uniform crossover as per our initial plan. Recreation of EPSO took more effort than was estimated because of the unknown variables present in the EPSO algorithm. Fortunately, these were estimated after testing the semi-complete algorithm on these variables. We extended EPSO by utilizing techniques from Evolutionary Algorithms in order to make EPSO more robust, and we succeeded in applying these techniques. The resulting algorithm showed improved results over the original, reaching better optima on average in an equal number of evaluations.

Furthermore we analyzed EPSO to examine the impact of the "foggy best-so-far region" on the results of the algorithm. With these tests we showed that EPSO relies on this fog factor more than it should, and we believe that EPSO attributes its success to the wrong parts of its mechanism. The original EPSO papers claim that their success lies in the application of mutation, however disabling the fog clearly shows that the fog is the main contributor to their results.

Finally we saw that our Combination algorithm also performed worse on average without this fog enabled, however the minima found were still impressive. This shows that our expansion with recombination is not the perfect answer either, and more research could to be done to make an expansion on Classic PSO that does not rely on the "foggy best-so-far region" for robust results. While relying on this fog is not inherently bad, it is not what these algorithms have set out to use.

6.1 FUTURE WORK

As with every project, there are a few stones here that could be turned over in the future. The short exploration of the Noise Dispersion parameter in Section 2.2 is one such example. While testing has taken place, more tests could be done to examine what is happening in more detail. Especially since the dependency of EPSO on the "foggy best-so-far region" is so immense, it is to be expected that further exploration of the Noise Dispersion parameter could give more insight.

More exploration on the Learning Dispersion parameter could be beneficial to the understanding of details of the EPSO algorithm. It could be interesting to see how much the parameter still influences EPSO even with the "foggy best-so-far region" disabled.

In the replication step of EPSO $r = 1$ is used, however having more mutated particles could prove beneficial to the exploration of the search space. While creating more particles would inherently create more evaluations, the benefits could outweigh the costs. For instance the Combination algorithm requires 3 times as many evaluations per iteration as the classic PSO does, changing r to 2 for EPSO would result in the same number of evaluations. Since the cloned particles are mutated individually they would explore more of the search space, and might improve performance enough to justify the increase in evaluations.

The crossover from the Combination algorithm is another aspect that could be explored in more detail. While uniform crossover is chosen because of its disruptive nature, 1-point or 2-point crossover could still prove very useful for certain test functions. It could be interesting to include the 'opposites' of each produced child, i.e. create a second child with the parameters from p_1 and p_2 that were unused by the first child. Creating more children would require more evaluations, and cause an increase in computing cost. Crossovers with more than 2 parents could also be explored, combining any number of particles to create offspring.

The focus of this thesis has been on recreating and expanding the EPSO algorithm, however the created Combination algorithm is not explored outside of the test functions yet. More time could be spent to apply the Combination algorithm on other complex problems. A basis for this could be applying the algorithm to NP-complete problems such as the knapsack or the traveling salesman problem [6]. Applying the Combination algorithm to real-world optimization problems could very well be a thesis on its own.

Part I

APPENDIX

A TEST FUNCTIONS

A common practice when creating optimization algorithms is to apply test functions for optimization to evaluate performance. These functions are often hard to solve without looking at the formula itself, simulating optimization on a problem with unknown constraints. For testing the EPSO algorithm we will use two variants of the Schaffer's function [13], the Rosenbrock function [16] and a Sphere function.

Schaffer's function F2[1]:

$$f(\vec{x}) = 0.5 + \frac{\sin^2\left(x_1^2 - x_2^2\right) - 0.5}{\left(1 + 0.001\left(x_1^2 + x_2^2\right)\right)^2} \tag{A.1}$$

Schaffer's function F6[2]:

$$f(\vec{x}) = 0.5 + \frac{\sin^2\left(\sqrt{x_1^2 + x_2^2}\right) - 0.5}{\left(1 + 0.001\left(x_1^2 + x_2^2\right)\right)^2} \tag{A.2}$$

Rosenbrock's function:

$$f(\vec{x}) = \sum_{i=1}^{n-1}\left(100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right) \tag{A.3}$$

Sphere function:

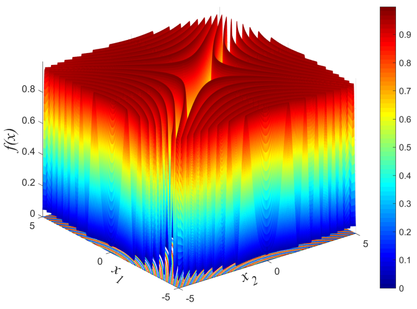
$$f(\vec{x}) = \sum_{i=1}^{n} x_i^2 \tag{A.4}$$

These functions provide a variety between extremely complex surfaces and relatively simple ones. In Table A.1 the domain of each of the test functions and the number of dimensions used is shown. The domain is later used to initialize the particles during testing. Figure A.1 shows a 3-dimensional plot of each test function using only 2 dimensions as input. For each of these functions lower values are better, with the global optimum being 0 for each function. The difficulty of the Schaffer functions is visible when looking at the graphs in Figure A.1. The Rosenbrock and Sphere functions seem less complicated visually, however using them on a tuple of 30 variables at once makes them quite complex.

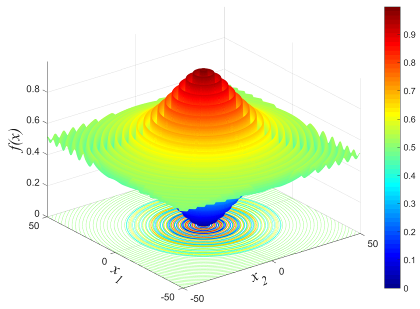
1 Sometimes referred to as 'Modified Schaffer's function #2'.
2 Often referred to as just 'Schaffer's function'.

Function	n	Domain
Schaffer F2	2	$[-100, 100]^n$
Schaffer F6	2	$[-100, 100]^n$
Rosenbrock	30	$[-15, 15]^n$
Sphere	30	$[-25, 25]^n$

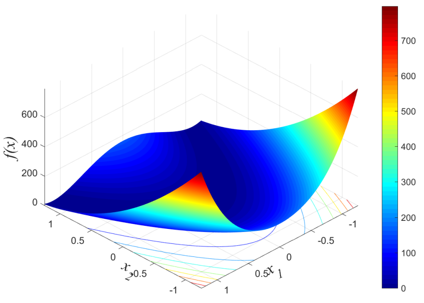
Table A.1: Parameters used in the test functions.



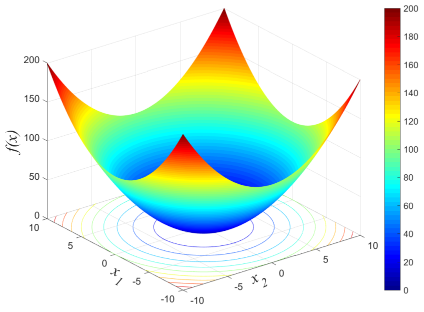
(a) Schaffer F2



(b) Schaffer F6



(c) Rosenbrock



(d) Sphere

Figure A.1: Three-dimensional plots of the test functions with $n = 2$ and domains set to illustrate the curves of the functions. Plots generated with MATLAB [9] code by Ali R. Alroomi [1]. The domain of each plot is set to illustrate the surface of the function, note that the Rosenbrock function is rotated 90°.

BIBLIOGRAPHY

[1] Ali R. Alroomi. *Power Systems and Evolutionary Algorithms - Unconstrained*. 2015. URL: http://al-roomi.org/benchmarks/unconstrained/.

[2] Hans-Georg Beyer. "Toward a theory of evolution strategies: On the benefits of sex — the $(\mu/\mu, \lambda)$ theory." In: *Evolutionary Computation* 3.1 (1995), pp. 81–111.

[3] Hans-Georg Beyer. "Toward a theory of evolution strategies: Self-adaptation." In: *Evolutionary Computation* 3.3 (1995), pp. 311–347.

[4] Russ C Eberhart and Yuhui Shi. "Comparing inertia weights and constriction factors in particle swarm optimization." In: *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*. Vol. 1. IEEE. 2000, pp. 84–88.

[5] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*. Vol. 53. Springer, 2003.

[6] Michael R Garey and David S Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. Vol. 29. wh freeman New York, 2002.

[7] Frank Heppner and Ulf Grenander. "A stochastic nonlinear model for coordinated bird flocks." In: *The ubiquity of chaos* (1990), pp. 233–238.

[8] James Kennedy. "Particle swarm optimization." In: *Encyclopedia of machine learning*. Springer, 1995, pp. 760–766.

[9] *MATLAB and Statistics Toolbox Release R2015b*. The Mathworks, Inc. Natick, Massachusetts, 2015.

[10] Vladimiro Miranda and Nuno Fonseca. "EPSO-best-of-two-worlds metaheuristic applied to power system problems." In: *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*. Vol. 2. IEEE. 2002, pp. 1080–1085.

[11] Vladimiro Miranda and Nuno Fonseca. "EPSO-evolutionary particle swarm optimization, a new algorithm with applications in power systems." In: *Transmission and Distribution Conference and Exhibition 2002: Asia Pacific. IEEE/PES*. Vol. 2. IEEE. 2002, pp. 745–750.

[12] Vladimiro Miranda and Nuno Fonseca. "New evolutionary particle swarm algorithm (EPSO) applied to voltage/VAR control." In: *Proc. 14th Power Syst. Comput. Conf.* 2002, p. 6.

[13] Sudhanshu K Mishra. "Some new test functions for global optimization and performance of repulsive particle swarm method." In: (2006).

[14] Riccardo Poli, James Kennedy, and Tim Blackwell. "Particle swarm optimization." In: *Swarm intelligence* 1.1 (2007), pp. 33–57.

[15] Craig W Reynolds. “Flocks, herds and schools: A distributed behavioral model.” In: *ACM SIGGRAPH computer graphics* 21.4 (1987), pp. 25–34.

[16] H. H. Rosenbrock. “An automatic method for finding the greatest or least value of a function.” In: *The Computer Journal* 3.3 (1960), pp. 175–184.

[17] William M Spears and Kenneth A De Jong. *An analysis of multi-point crossover*. Tech. rep. Naval Research Lab Washington DC, 1990.

[18] William M Spears and Kenneth A De Jong. *On the virtues of parameterized uniform crossover*. Tech. rep. Naval Research Lab Washington DC, 1995.

[19] *The R Project for Statistical Computing*. The R Foundation. 2017. URL: `https://www.r-project.org/`.