Parameter Sensitivity Patterns in the Plant Propagation Algorithm

Marleen de Jonge[a] and Daan van den Berg[b]
Informatics Institute, University of Amsterdam, The Netherlands

Keywords: Plant Propagation Algorithm, Evolutionary Algorithms, Parameterization, Parameter Sensitivity, Parameter Robustness, Offspring Number, Population Size, Metaheuristics, Function Optimization.

Abstract: The parameter sensitivity of the plant propagation algorithm's performance, defined as the maximum impact on attained objective values, is studied on function optimization. The number of offspring and population size are varied across a large range of values and tested on multidimensional benchmark test functions. As the dimensionality of a function increases, the parametric sensitivity shows one of three distinct different patterns: sublinear increase, superlinear increase or an up-down phase transition. We conjecture that the difference in algorithmic behaviour might be due to the intrinsic mathematical properties of the functions themselves.

1 INTRODUCTION

Hard optimization problems come in a broad bouquet of both theoretical and real-world applications. Continuous problems such as molecular docking, scientific imaging techniques and process flowsheet optimization in chemical plants (Meier et al., 2010)(Chambolle and Pock, 2016)(Salhi and Fraga, 2011) are countered by discrete exemplars such as vehicle routing, protein design and computational creativity (Kara et al., 2007)(Allouche et al., 2014)(Paauw and Van den Berg, 2019). To date, techniques to obtain good solutions for such problems can roughly be divided into two categories: *exact* methods and *heuristic* methods(Puchinger and Raidl, 2005)[1]. Exact methods exhaustively search the complete or partially pruned state space, thereby guaranteeing to find an optimal solution, but runtimes tend to increase exponentially (or worse) with problem instance size, prohibiting practical deployment on any real-world problem. By sacrificing the guarantee of optimality, heuristic methods produce solutions which are good (enough), and reachable within reasonable time budgets (Puchinger and Raidl, 2005). For this reason, the development (meta)heuristic algorithms has flourished as an approach for a multitude of optimization tasks.

All is not rosegardens however. As the field has witnessed a blossoming proliferation of new methods, the often creative metaphorical references (to nature) obfuscate real progress in the field and prevent transparency, replicability and scientific rigor (Sörensen, 2015). Furthermore, the community has been complaining for ages that the wildgrowth of new algorithms is seldomly parameterically tested, and the performance across different problems is poorly understood (Sörensen, 2015)(Eiben and Smit, 2011). As new research on heuristic methods seems primarily focused on achieving slight improvements on a single problem instance, in-depth understanding of algorithmic behaviour and performance dependencies obtain much less attention. Kenneth Sörensen has in fact devoted a whole paper to it, rightfully criticizing this development as it "threatens to come at the expense of scientific rigor in the field of metaheuristics" (Sörensen, 2015). The authors of this study generally agree to these viewpoints.

Extensive understanding of parameter settings and their contribution to algorithmic performance can be of great importance, both in terms of its best found objective value (or *fitness*) and in terms of its robustness (Smit and Eiben, 2009). Unfortunately, the vast number of possible parameter configurations, the complexity of their interaction and the diversity of measuring methods make the process of finding optimal parameter settings often harder than disentangling a wall of brambles.

One relatively successful algorithm lacking a parameterization study is the plant propagation algo-

[a] https://orcid.org/0000-0003-4911-2647
[b] https://orcid.org/0000-0001-5060-3342

[1] For some spun out examples, see (Sleegers et al., 2020) and (van den Berg et al., 2016).

de Jonge, M. and van den Berg, D.
Parameter Sensitivity Patterns in the Plant Propagation Algorithm.
DOI: 10.5220/0010134300920099
In *Proceedings of the 12th International Joint Conference on Computational Intelligence (IJCCI 2020)*, pages 92-99
ISBN: 978-989-758-475-6

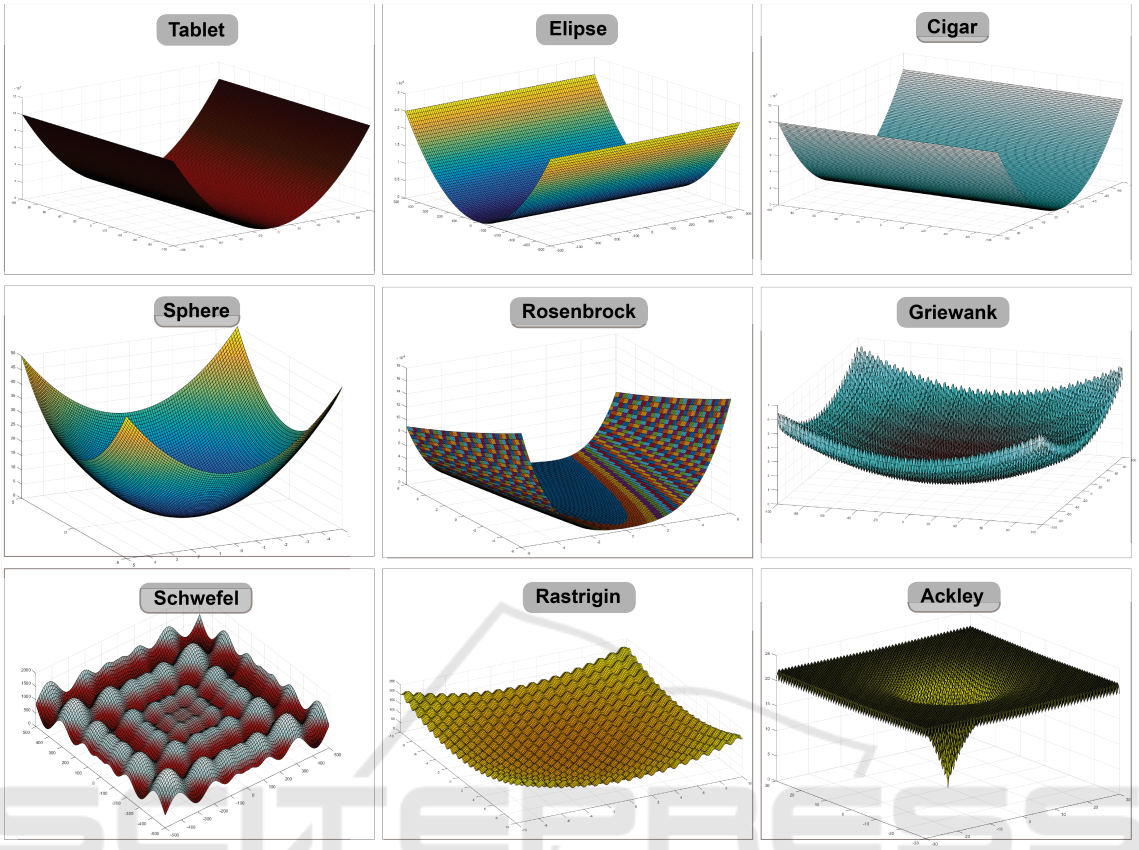

Figure 1: Nine multidimensional benchmark test functions used in this study. All are continuous and dimensionally scalable, but vary in properties such as modality, range, domain, and elementary mathematical constituents. Shown here are the 2D versions.

rithm (PPA), introduced by Abdellah Salhi and Eric S. Fraga (Salhi and Fraga, 2011). Even though it *is* metaphorically linked to nature, its paradigm is rooted in simple operations, and its parametric canopy is quite modest. It has been demonstrated to work well on a variety of continuous benchmark test functions as well as on some discrete problems such as the Traveling Salesman Problem, University Timetabling and even explicitly tracking down NP-complete problem instances (Sulaiman et al., 2014)(Salhi and Fraga, 2011) (Selamoğlu and Salhi, 2016)(Vrielink and van den Berg, 2019) (Geleijn et al., 2019)(Sleegers and van den Berg, 2020). Additionally, the availability of open source python repositories lower the threshold for implementation, experimentation, testing, and replication(De Jonge, 2020).

In most experimentation on PPA, its parameters are arbitrarily set, and little is known about their individual or interdependent impact on performance. Following a preliminary investigation on two-dimensional functions(de Jonge and van den Berg, 2020), we will address multidimensional functions, which behave substantially different. We investigate the interplay of two interdependent parameters of the algorithm, the population size (*popSize*) and the maximum number of offspring per individual (n_{max}). We explore 400 parameter settings of PPA on nine carefully chosen n-dimensional benchmark test functions. Adopting a practical approach advocated by Selmar Smit and Gusz Eiben, we choose parameter values that optimize the algorithm's performance while simultaneously mapping *how* its performance depends on its parameter settings, referring to the latter as 'robustness' or 'sensitivity'. The setup will yield some puzzling results, and raise questions about the interplay between the algorithm and the problem instance it is trying to solve[2]. Luckily, we also get some straight up answers.

The rest of the paper is organized as follows: the following section will provide an explanation on PPA, whereas Section 3 will introduce the benchmark test

[2]An elaborate discussion on exactly this topic took place in the GECCO-workshop "Good Benchmarking Practices for Evolutionary Computation", which took place (online) at GECCO2020, and also supplied a public paper (Bartz-Beielstein et al., 2020).

functions used in this study. Thereafter, Section 4 offers a detailed description of the experiment after which the results and conclusions will be presented in Section 5. We finalize the paper by discussing the choices we made along the way, and the shortcomings our approach might have in Section 6.

2 PLANT PROPAGATION ALGORITHM

In its seminal form, PPA is an implementation on continuous optimization problems (Algorithm 1) (Salhi and Fraga, 2011), and the enveloping paper also included some runs on a suite of benchmark test functions, all of which have been included in this study (Figure 1, Table 1). The only difference is that for this investigation, all global minima are normalized to zero by applying the necessary 'vertical' shifts.

Algorithm 1: PPA pseudo code.

Require: objective *f(x), x* $\in \mathcal{R}$
1: Generate population $P = \{p_1, p_2, ..., p_{popSize}\}$
2: $g \leftarrow 1$
3: **for** $g \leftarrow 1$ **to** g_{max} **do**
4: compute $N_i = f(p_i), \forall p_i \in P$
5: sort P in descending order of N_i
6: create new population ϕ
7: **for** each $p_i \in P$ **do** {take best *popSize* only}
8: $r_i \leftarrow$ set of offspring where both the size of the set and the mutability for each new individual is functionally related to the fitness N_i
9: $\phi \leftarrow \phi \cup r_i$ merge population with offspring; death occurs by selection of *popSize* best individuals
10: **end for**$P \leftarrow \phi$ new population
11: **end for**
12: **return** P, population of solutions

In this population based algorithm, an individual is a vector of D points, in which D is the dimensionality of the benchmark function at hand, and its objective value is calculated from the dimensionalized benchmark function on the vector. After randomly initializing *popSize* individuals, calculating objective values and normalizing them to [0, 1] (Eq. 1), the hyperbolic tangent function maps the normalized fitness N_i nonlinearly to (0,1) (Eq. 2). This asymptotic function serves to prevent offspring to be generated at the exact same location as the parent, and "provide[s] a means of emphasising further better solutions over those which are not as good" (Salhi and Fraga, 2011). Then, each generation, individuals within the population with higher mapped fitness N_i will generate more offspring, whereas individuals with the lower N_i will generate fewer offspring (Eq. 3). But whereas the number of offspring n_i is proportional to N_i, the mutability m_i of an individual's offspring is *inversely* proportional to the fitness (Eq. 4).

$$z(x_i) = \frac{f(x_{max}) - f(x_i)}{f(x_{max}) - f(x_{min})} \tag{1}$$

$$N_i = \frac{1}{2}(tanh(4z(x_i) - 2) + 1) \tag{2}$$

$$n_i = [n_{max} * N_i * r] \tag{3}$$

$$m_i = 2(1 - N_i)(r - 0.5) \tag{4}$$

In these equations, r is random number $\in [0, 1)$, drawn anew every time it is used, and m_i is applied to every dimension of the benchmark function at hand. The newly created offspring is then added tot the population, which is sorted to fitness, and the *popSize* best individuals are selected for the next generation. Note that thereby, PPA always retains its best individual, and adopts an *elitist* approach – its best fitness cannot decrease during a run.

3 BENCHMARK FUNCTIONS

Benchmarking is a relatively standardized way of measuring performance and behaviour of algorithms (Jamil and Yang, 2013). Literature on test functions however, is quite diverse and sometimes poorly defined, causing multiple function definitions and specifications to circulate. For this reason, we explicit list the used formulae (Table 1) and supply publicly accessible source code (De Jonge, 2020). The suite was compiled by Wouter Vrielink and forms a union of functions earlier used on the fireworks algorithm or the plant propagation algorithm (Tan and Zhu, 2010)(Salhi and Fraga, 2011)(Vrielink and van den Berg, 2019). We use the identical multi-dimensional test functions from Vrielink's suite to enable direct comparison between all forementioned studies. All code is written in Python version 3.6.3 and can be found in a publicly accessible online repository(De Jonge, 2020).

Table 1: The entire test suite of multidimensional benchmark functions used in this study. Test have been performed on six different dimensionalities with 400 parameter settings on each dimensionality.

Name	Function	Bounds
Tablet	$f(\mathbf{x}) = 10000x_1^2 + \sum_{i=2}^{D} x_i^2$	$x \in [-100, 100]^D$
Ellipse	$f(\mathbf{x}) = \sum_{i=1}^{D} 10000^{(\frac{i-1}{D-1})} x_i^2$	$x \in [-100, 100]^D$
Cigar	$f(\mathbf{x}) = x_1^2 + \sum_{i=2}^{D} 10000x_i^2$	$x \in [-100, 100]^D$
Sphere	$f(\mathbf{x}) = \sum_{i=1}^{D} x_i^2$	$x \in [-100, 100]^D$
Rosenbrock	$f(\mathbf{x}) = \sum_{i=1}^{D-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	$x \in [-5, 10]^D$
Griewank	$f(\mathbf{x}) = 1 + \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} cos\frac{x_i}{\sqrt{i}}$	$x \in [-600, 600]^D$
Schwefel	$f(\mathbf{x}) = 418.9829D - \sum_{i=1}^{D} (x_i \cdot sin(\lvert\sqrt{x_i}\rvert))$	$x \in [-500, 500]^D$
Rastrigin	$f(\mathbf{x}) = 10D + \sum_{i=1}^{D} (x_i^2 - 10cos(2\pi x_i))$	$x \in [-5.12, 5.12]^D$
Ackley	$f(\mathbf{x}) = -20 \cdot exp(-0.2 \cdot \sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}) - exp(\frac{1}{D}\sum_{i=1}^{D} cos(2\pi x_i)) + 20 + e$	$x \in [-100, 100]^D$

4 EXPERIMENT

The experimental setup is as massive as it is straightforward; we perform optimization runs with PPA on nine different n-dimensional benchmark functions (Table 1) in six dimensionalities $D \in \{2, 5, 10, 20, 50, 100\}$ resulting in 54 heatmaps (see Figure 2). Every heatmap is comprised of 400 cells for 400 different parameter settings $1 \leq popSize \leq 40$, $1 \leq n_{max} \leq 10$, and every cell contains ten PPA-runs of 10,000 function evaluations, adding up to a grand total of roughly 2.16 billion function evaluations for the whole experiment. Due to parallelization efforts however, the runtime for the experiment could be limited to approximately 34 hours on 10 nodes with 12 to 48 cores each, SURFSara's LISA cluster computer at Amsterdam Science Park.

After completing ten runs for one parameter setting, the median was taken to a heatmap's cell (e.g. Rosenbrock's six heatmaps in Figure 2). Generally speaking, but surely in Rosenbrocks case, the average error increases with the dimensionality of the benchmark function, resulting in ever darker heatmaps. But this is to be expected; for all benchmark test functions used in this study, the dimensionality is reflected in the sum over its dimensions ($\sum_{i=1}^{D}$), whose terms often cannot be negative within the domain. Therefore, the randomly chosen initial values are much higher on the 50-dimensional function than on the same function in 10 dimensions.

So how to account for this dimensional upscaling when quantifying the parametric influence on the attained objective value? Considering these observations, the degree to which PPA is susceptible to changes in parameter values can be seen as the maximal difference its various parameterizations might have within the 10,000 evaluation range of this experiment. We therefore choose to define the parameter sensitivity as the interval between the best and worst heatmap cell relative to the range in the attained objective value:

$$\frac{Max(f_{popSize,n_{max}}(x)) - Min(f_{popSize,n_{max}}(x))}{\mu_{benchmark}} \quad (5)$$

Finally, for $\mu_{benchmark}$, we take the mean of 10,000 randomly sampled points (see Figure 3). This value is alone sufficient for these functions, as the global minimum of any of these in any dimensionality is zero.

5 RESULTS & CONCLUSION

From a simple and straightforward experimental setup come some surprising results. For all functions, performance of the plant propagation algorithm deteriorated with dimensionality, comparable to earlier results (Vrielink and van den Berg, 2019). The exception is the Ackley function, on which the found minimum is near 20 for all $D > 2$, but this too is known from literature (although the dimensionality

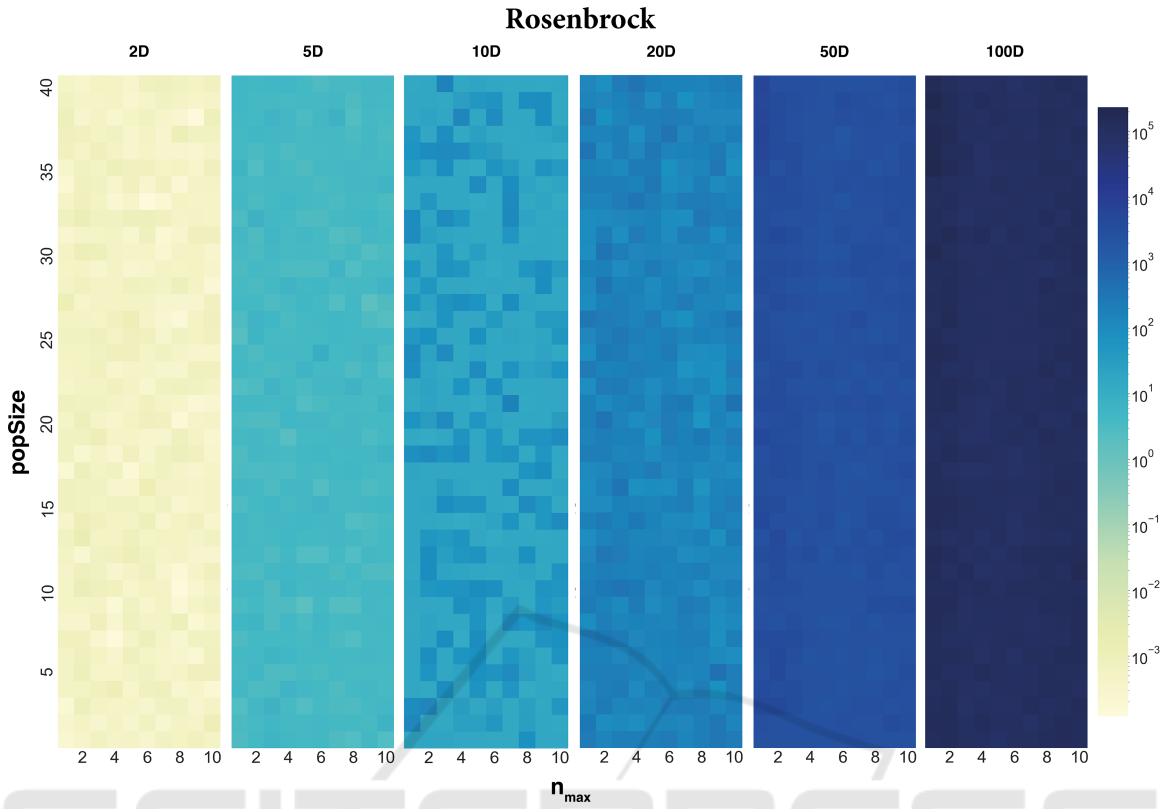

Figure 2: Performance of the plant propagation algorithm with 400 different parameter settings on the Rosenbrock function. Every cell contains the median value of ten runs of 10,000 evaluations. As the dimensionality increases, so does the mean error ($\mu = 0.0005$ to $\mu = 112011$) and the variance thereof ($\sigma = 0.0002$ to $\sigma = 21744$). The absolute difference between the best and worst parameter settings increases faster than its mean error, signifying an increase of parameter sensitivity in D.

bound might be slightly more accurate in (Vrielink and van den Berg, 2019)). The parameter settings, in all cases except the last three, did not make a difference of more than a magnitude.

A second, more interesting pattern, can be observed by assessing the absolute parameter sensitivities in respect to the dimensionalities. For two benchmark test functions, the Tablet and the Ellipse, the parameter sensitivity apparently shows a **sublinear increase** with the dimensionality, which could possibly be characterized by a polynomial of $O(n^c)$ with $0 < c < 1$. For four others, the Cigar, Sphere, Rosenbrock and Griewank functions, the parameter sensitivity displays a **superlinear increase** in dimensionality, which might be best characterized by a polynomial of $O(n^c)$ with $c > 1$. In the three last functions, Schwefel, Rastrigin and Ackley, the parameter sensitivity shows something that looks like a **up-down phase transition**. A dramatic initial increase in sensitivity reaches an early summit somewhere between 5 and 10 dimensions, after which a rapidly collapse follows, in one case even to near zero.

But a third, perhaps even more intriguing pattern, can be discerned from ordering the nine sensitivity patterns (Figure 4). With some imagination, one can see the dimensional parameter sensitivity patterns undergoing a structural phase transition *themselves*, from sublinear, to superlinear, to up-down. When looking closely, the up-down phase transition pattern might already be materializing in the Griewank function, perhaps even in the Rosenbrock. Going back to Table 1, one could easily wonder whether it is a coincidence that the 'simple quadratics' only show sublinear and superlinear sensitivity patterns (the table is sorted in the same order as Figure 4). The second category, 'quadratic plus trigonometric' shows only superlinear or up-down phase transition patterns. The third category, 'trigonometric plus exponential' only has an up-down phase transition pattern, and it is hard not to conjecture that these functional definitions *themselves* shelter the structural properties that determine the parameter sensitivities of PPA, and perhaps other (population based) heuristic algorithms. An inquiry into what property constitutes which pattern could be extremely valuable. And computationally expensive, not to mention.

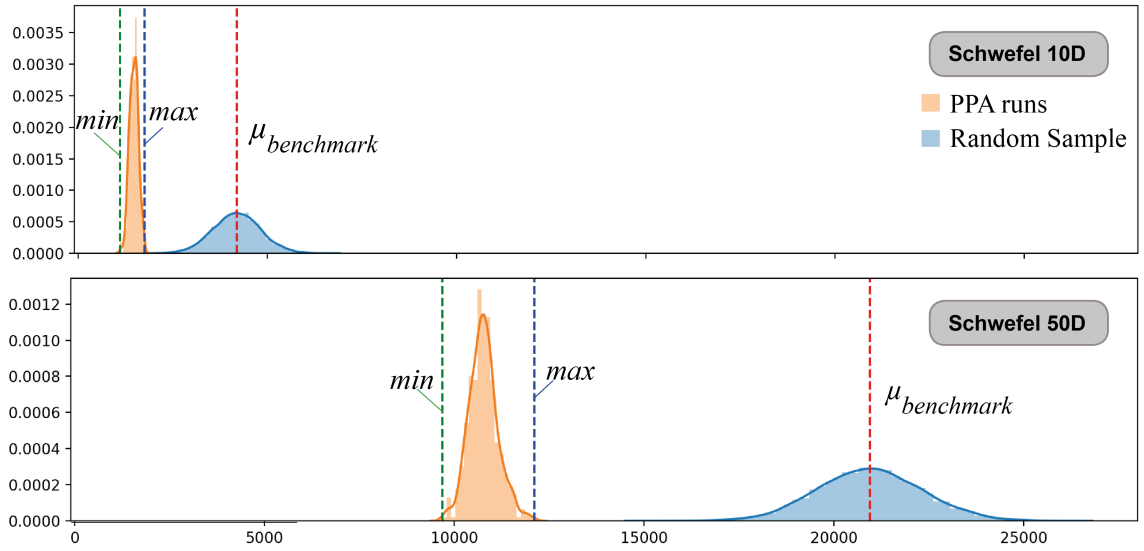

Figure 3: Parameter sensitivity is defined as the difference between the absolute best parameterization and absolute worst parameterization relative to the average initial condition, which, for most benchmark functions, increases with the dimensionality of the function. PPA's senstivity is 0.16 on the Schwefel 10D function, but 0.12 on the Schwefel 50D function, a counterintuitive result which can be seen in more detail in Figure 4.

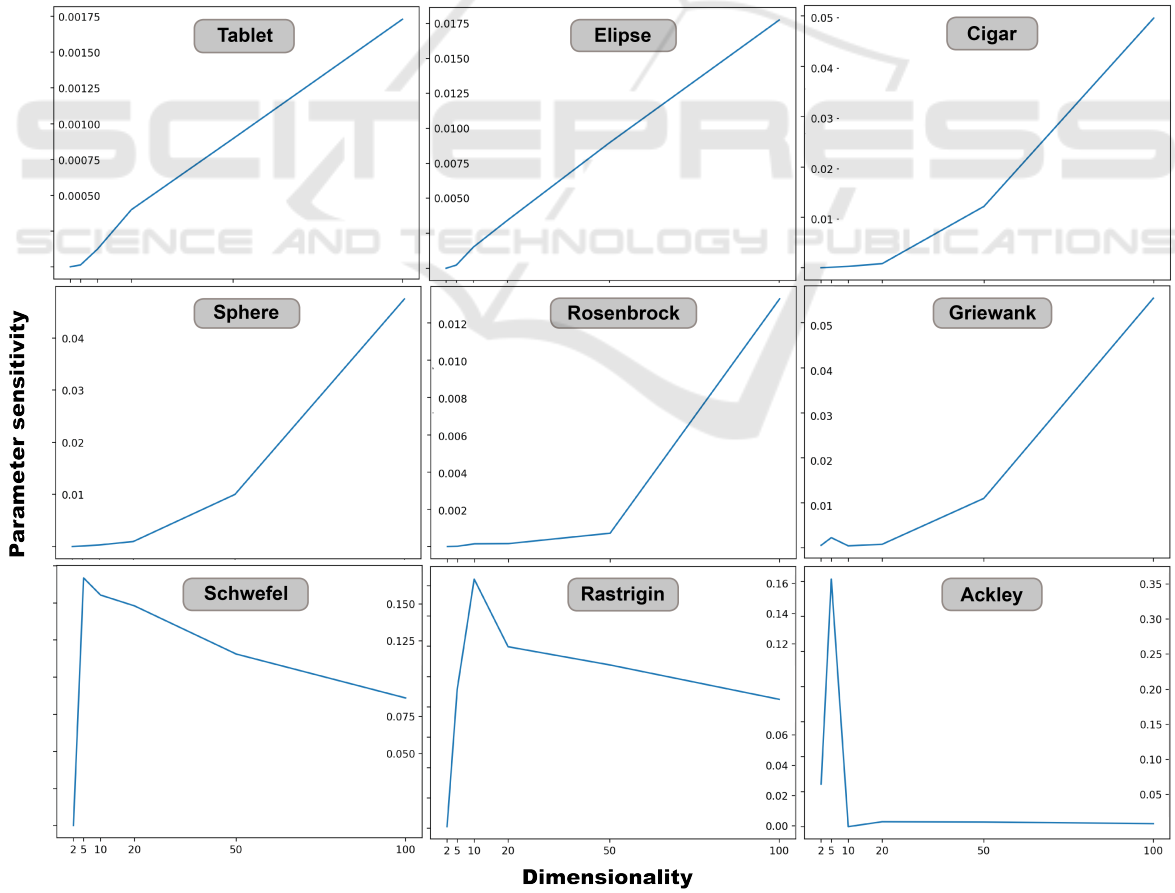

Figure 4: Parameter sensitivity patterns for PPA on benchmark test functions comes in three types: sublinear increase, superlinear increase and up-down phase transition. A pattern could be to some degree related to a function's mathematical properties, such as being quadratic-only, quadratic-trigonometric, or exponential.

6 DISCUSSION

From a simple and straightforward experimental came some not-so-very-simple-and-straightforward answers. Along the way, we made some choices that are well open for discussion. First of all, we studied just two interdependent PPA-parameters and there is absolutely no reason to presume that these results are stable under replacement of the *tanh*-function, different mutability rates, or a different population selection procedure. Second, the choice of benchmark test functions could matter. Does a 'combination of Schwefel and Rastrigin' show an intermittent stability pattern? Third, the measure of sensitivity could be different, based on standard deviations, or based on the number of evaluations required to reach a certain target value v, as suggested in (Eiben and Smit, 2011). It could also be seen as a measure of uncertainty relative to a run's performance, which is in turn relative to the budget of function evaluations (set to 10,000 in this study). Fourth, considering the difference in sensitivity patterns for continuous benchmark functions, bifurcated study into more real-life examples such as the optimization of chemical plant parameterizations might provide useful insights. First, the parameter sensitivity could be assessed, similar to the study presented here. Second, an algebraic comparison between it parameterization-fitness projection and the known benchmark functions could be made to further our knowledge of the relation between parameter sensitivity and the properties of the continuity it is trying to minimize. Last but not least, the best-to-worst order of individual heatmap cells could be studied; is there a pattern to be found, even if just modest? All in all, many possible roads lead into the future, and we should make efforts to progress pedestrianally[3].

ACKNOWLEDGEMENTS

Reitze Jansen (UvA), meticulous and accurate as ever, was kind enough to pull some mistakes from this paper. Thanks Reitze.

REFERENCES

Allouche, D., André, I., Barbe, S., Davies, J., de Givry, S., Katsirelos, G., O'Sullivan, B., Prestwich, S., Schiex, T., and Traoré, S. (2014). Computational protein design as an optimization problem. *Artificial Intelligence*, 212:59–79.

Bartz-Beielstein, T., Doerr, C., Bossek, J., Chandrasekaran, S., Eftimov, T., Fischbach, A., Kerschke, P., Lopez-Ibanez, M., Malan, K. M., Moore, J. H., Naujoks, B., Orzechowski, P., Volz, V., Wagner, M., and Weise, T. (2020). Benchmarking in optimization: Best practice and open issues.

Chambolle, A. and Pock, T. (2016). An introduction to continuous optimization for imaging. *Acta Numerica*, 25:161–319.

De Jonge, M. (2020). Source code of algorithm used in the experiment. https://github.com/marleentheyoung/BScKI_Thesis_PPA.

de Jonge, M. and van den Berg, D. (2020). Plant propagation parameterization: Offspring & population size. *Evo* 2020*, page 19.

Eiben, A. E. and Smit, S. K. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31.

Geleijn, R., van der Meer, M., van der Post, Q., and van den Berg, D. (2019). The plant propagation algorithm on timetables: First results. *EVO* 2019*, page 2.

Jamil, M. and Yang, X.-S. (2013). A literature survey of benchmark functions for global optimization problems. *arXiv preprint arXiv:1308.4008*.

Kara, I., Kara, B. Y., and Yetis, M. K. (2007). Energy minimizing vehicle routing problem. In *International Conference on Combinatorial Optimization and Applications*, pages 62–71. Springer.

Meier, R., Pippel, M., Brandt, F., Sippl, W., and Baldauf, C. (2010). Paradocks: a framework for molecular docking with population-based metaheuristics. *Journal of chemical information and modeling*, 50(5):879–889.

Paauw, M. and Van den Berg, D. (2019). Paintings, polygons and plant propagation. In *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*, pages 84–97. Springer.

Puchinger, J. and Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *International work-conference on the interplay between natural and artificial computation*, pages 41–53. Springer.

Salhi, A. and Fraga, E. S. (2011). Nature-inspired optimisation approaches and the new plant propagation algorithm.

Selamoğlu, B. İ. and Salhi, A. (2016). The plant propagation algorithm for discrete optimisation: The case of the travelling salesman problem. In *Nature-inspired computation in engineering*, pages 43–61. Springer.

Sleegers, J. and van den Berg, D. (2020). Plant propagation & hard hamiltonian graphs. *Evo* 2020*, page 10.

Smit, S. K. and Eiben, A. E. (2009). Comparing parameter tuning methods for evolutionary algorithms. In *2009 IEEE congress on evolutionary computation*, pages 399–406. IEEE.

[3]Abdellian saying from 'The Book of Strawberries'; "Pedestrianally" (adj., adv.): calmly, considerately, unorganizedly, chaotically, improvisingly. Wisely taking things one step at the time; seeing where the road leads as events unfold.

Sörensen, K. (2015). Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, 22(1):3–18.

Sulaiman, M., Salhi, A., Selamoglu, B. I., and Kirikchi, O. B. (2014). A plant propagation algorithm for constrained engineering optimisation problems. *Mathematical problems in engineering*, 2014.

Tan, Y. and Zhu, Y. (2010). Fireworks algorithm for optimization. In *International conference in swarm intelligence*, pages 355–364. Springer.

van den Berg, D., Braam, F., Moes, M., Suilen, E., and Bhulai, S. (2016). Almost squares in almost squares: Solving the final instance. DATA ANALYTICS 2016, page 81.

Vrielink, W. and van den Berg, D. (2019). Fireworks algorithm versus plant propagation algorithm.