Nature inspired methods for optimization

A Julia primer for process engineering

Eric S. Fraga

Sargent Centre for Process Systems Engineering
Department of Chemical Engineering
University College London (UCL)

Dedicated to Jennifer and Sebastian, with love.

DOI 10.5281/zenodo.7016482

First edition 2022-08-19

Current version 2022-11-09 1a40907

Preface

Nature inspired methods for optimization have been available for many years. They are attractive because of their inspiration and because they are comparatively easy to implement. This book aims to illustrate this, using a relatively new language, Julia. Julia is a language designed for efficiency. Quoting from the developers of this language:

> Julia combines expertise from the diverse fields of computer science and computational science to create a new approach to numerical computing. [Bezanson et al. 2017]

The case studies, from industrial engineering with a focus on process engineering, have been fully implemented within the book, bar one example which uses external codes. All the code is available for readers to try and adapt for their particular applications.

This book does not present *state of the art* research outcomes. It is primarily intended to demonstrate that simple optimization methods are able to solve complex problems in engineering. As such, the intended audience will include students at the Masters or Doctoral level in a wide range of research areas, not just engineering, and researchers or industrial practitioners wishing to learn more about Julia and nature inspired methods.

Contents

One

Introduction

1.1 Optimization

The problems considered in this book will all be formulated as the minimization of the desired objectives. There may be one or multiple objectives. In either case, the general formulation is

$$\begin{aligned} \min_{d \in \mathcal{D}} z &= f(d) \\ g(d) &\leq 0 \\ h(d) &= 0 \end{aligned} \tag{1.1}$$

where d are the *decision* (or *design*) variables and $\mathcal{D}$ is the search domain for these variables. $\mathcal{D}$ might be, for instance, defined by a hyper-box in n dimensional space,

$$\mathcal{D} \equiv [a, b]^n \subset \mathbb{R}^n$$

when only continuous real-valued variables are considered. More generally, the problem may include integer decision variables. Further, the constraints, $g(d)$ and $h(d)$, will constrain the *feasible* points within that domain.

1.2 Process systems engineering

The focus of this book is on solving problems that arise in process systems engineering (PSE). Optimization plays a crucial part in many PSE activities, including process design and operation. The problems that arise may have one or more of these challenges, in no particular order:

- nonlinear models
- multi-modal objective function
- nonsmooth and discontinuous
- distributed quantities
- differential equations
- small feasible regions
- multiple objectives

Each of these aspects, individually, can prove challenging for many optimization methods, especially those based on gradients to direct the search for good solutions. Some problems may have several of these characteristics.

Three chapters will present example design problems which exhibit some of the above aspects:

1. purification section for a chlorobenzene production process: discontinuous, non-smooth, small feasible region;
2. design of a plug flow reactor: continuous distributed design variable with behaviour described by differential equations;
3. heat exchanger network design: combinatorial, non-smooth, complex problem formulation.

These problems may be challenging for gradient based methods and therefore motivate the use of meta-heuristic methods. In this book, I will present meta-heuristic methods inspired by nature.

1.3 Nature inspired optimization

There are many optimization methods and different ways of classifying these methods. One classification used often is *deterministic* versus *stochastic*. The former include direct search methods [Kelley 1999] and what are often described as mathematical programming, such as the Simplex method [Dantzig 1982] for linear problems and many different methods for nonlinear problems [Floudas 1995]. The main advantage of deterministic methods is that they will obtain consistent results for the same starting conditions and may provide guarantees on the quality of the solution obtained and/or the performance of the method, such as speed of convergence to an optimal solution. If you are interested in using Julia for mathematical programming, the JuMP[1] package is popular.

Stochastic methods, as the name implies, are based on random behaviour. The main implication is that the results obtained will vary from one attempt to another, subject to the random number generator used. However, the advantage of stochastic methods can be the ability to avoid getting stuck in local optima and potentially find the global optimum or optima for the problem.

Some popular stochastic methods include *simulated annealing* [Kirkpatrick et al. 1983] and *genetic algorithms* [Holland 1975] but many others exist [Lara-Montaño et al. 2022]. Simulated annealing, as the name implies, is based on the process of annealing, typically in the context of the controlled cooling of liquid metals to achieve specific properties (hardness, for instance). Genetic algorithms mimic Darwinian evolution, for instance, by using *survival of the fittest* to propagate good solutions and evolve better solutions. One common aspect of many stochastic methods is that they are inspired by nature.

In this book, we will consider three such nature inspired optimization methods: genetic algorithms, particle swarm optimization [Kennedy and Eberhart 1995], and a plant propagation algorithm [Salhi and Fraga 2011]. These are chosen somewhat arbitrarily as examples of stochastic methods. The selection is in no way intended to indicate that these are the best meth-

[1] https://jump.dev/

ods for any particular problem. The aim is to illustrate the potential of these methods, how they may be implemented, and how they may be used to solve problems that arise in process systems engineering.

1.4 Literate programming

This book includes all the code implementing the nature inspired optimization methods and most of the case studies, sufficient to allow readers to attempt the problems themselves.

> Programming is at heart a practical art in which real things are built, and a real implementation thus has to exist. [Kay 1993]

The book has been written using *literate programming* [Knuth 1984]. The motivation for literate programming, to quote Donald Knuth, comes from:

> [...] the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be *works of literature*. [Knuth 1984]

Therefore, this book may be considered to be the documentation of the code used to implement the methods **and** to evaluate the problems defined in the case studies. The code presented in the book is automatically exported to the source code files, suitable for immediate invocation.

The technology supporting literate programming in this case is `org mode`[2] (version 9.5.4). `Org mode` is a special mode in the Emacs editor[3] (version 29.0.50.0). `Org mode` supports *code blocks* which may include programming code in a very wide range of languages. These code blocks are *tangled* into code files [Schulte and Davison 2011]. In the case of this book, all the code can be found in the author's `github` repository[4]. As well as enabling tangling to create the source code files, `org`

[2] `https://orgmode.org/`
[3] `https://www.gnu.org/software/emacs/`
[4] `https://github.com/ericsfraga/NatureInspiredOptimization.jl`

mode supports exporting documents to a variety of different targets, including PDF, epub, and HTML. The HTML version of the book is freely available[5].

Lastly, the literate programming support in org mode not only enables tangling, it also allows for code to be run directly within the Emacs editor while editing the document, with the results of the evaluation inserted into the document [Schulte and Davison 2011]. This book makes full use of this capability to process the results of the optimization problems, including extracting and generating statistical data with awk, grep, and similar tools, and plotting the outcomes with gnuplot[6].

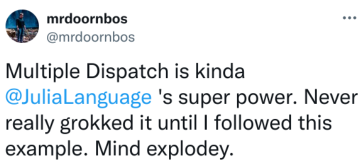
1.5 The Julia language

Julia[7] is a multi-purpose programming language with features ideally suited for writing generic optimization methods and numerical algorithms in general. To repeat the quote from the initial authors of the language,

> Julia combines expertise from the diverse fields of computer science and computational science to create a new approach to numerical computing.[Bezanson et al. 2017]

For the purposes of this book, the key features of Julia are the following:

- dynamic typing for high level coding;
- multiple dispatch to create generic code that may be easily extended;



[5] http://www.ucl.ac.uk/~ucecesf/niobook
[6] http://www.gnuplot.info/
[7] https://julialang.org/

[tweet][8]

- functional programming elements for operating on data easily;
- integrated package management system to enable re-use and distribution; and,
- the REPL (*read-evaluate-print* loop) for exploring the language and testing code.

In this section, some of these aspects will be illustrated through code elements that will be used by all the nature inspired optimization methods presented in this book.

Julia version 1.7 has been used for all the codes in this book. I have tried to ensure that the style guide[9] for writing Julia code has been followed throughout.

The package and modules

The codes presented in this book are tangled into a Julia package called `NatureInspiredOptimization` and available from the author's `github` repository[10]. This package can be added to your own Julia installation by entering the package manager system in Julia using the `]` key and then using the `add` command with the URL of the `git` repository for the package. This will add not only the package specified but also any dependencies, i.e. other packages, specified by this package. Hitting the backspace key will exit the package manager. Once added, the package and any sub-packages it may define, can be accessed through the `using` Julia statement. This will be illustrated in the examples in this book.

[8] https://twitter.com/mrdoornbos/status/1557332205709541383?s=20&t=Raxk1hSjWBmrgqdvU_VIVw

[9] https://docs.julialang.org/en/v1/manual/style-guide/

[10] https://github.com/ericsfraga/NatureInspiredOptimization.jl

Some external packages have been used in writing the code presented in the book. These include `DifferentialEquations`[11], `Plots`[12], `JavaCall`[13], and `Printf`[14]. These will automatically be installed when installing the `NatureInspiredOptimization` package. The one exception is the `Jacaranda` package (see Section 3.2 below), written by the author and not currently registered in the Julia Registry. Therefore, this package needs to be added explicitly.

In summary, installing the `NatureInspiredOptimization` package can be achieved as follows:

```
$ julia
julia> ]
pkg> add https://gitlab.com/ericsfraga/Jacaranda.jl
pkg> add https://github.com/ericsfraga/NatureInspiredOptimization.jl
[...]
(@v1.7) pkg> BACKSPACE
julia> ^d
$
```

Note that the `Jacaranda` package is found in a `gitlab` repository while the `NatureInspiredOptimization` package is on `github`.

Objective function

All optimization problems in this book will define an objective function. The methods implemented will all be based on the same *signature* for the objective function:

```
(z, g) = f(x; π)
```

where *f* is the name of the function implementing the objective function for the optimization problem, to be evaluated at the point x in the search space.

---

[11] https://diffeq.sciml.ai/stable/
[12] https://docs.juliaplots.org/latest/
[13] https://juliainterop.github.io/JavaCall.jl/
[14] https://docs.julialang.org/en/v1/stdlib/Printf/

The second optional argument, π, will consist of a data structure of any type which includes parameters that are problem dependent. The function f is expected to return a *tuple*. The first entry in the tuple, z, is the value of the objective function for single objective problems and a vector of values for multi-objective problems. The second element of the tuple, g, is a single real number which indicates the feasibility of the point x: $g \leq 0$ means the point is feasible; $g > 0$ indicates an infeasible point, with the magnitude of g ideally representing the amount of constraint violation, when this is possible. In determining the *fitness* of a point, see Section 2.2 below, both the value(s) of z and g will be used.

To allow for both single and multi-objective problems in the code that follows, the generic comparison operators ≻ (`\succ` in Julia) and ≽ (`\succeq` in Julia) will be used. $a \succ b$ means that a is *better than* b and $a \succeq b$ means that a is *at least as good as* b. For single objective **minimization** problems, which will be the case for the case studies presented later in the book, these operators correspond to the *less than* (<) and *less than or equals* (≤) comparisons:

```
≻(a :: Number, b :: Number) = a < b
≽(a :: Number, b :: Number) = a ≤ b
export ≻, ≽
```

This code segment illustrates several Julia features:

1. the single line definition of a function using assignment;

2. the definition of a binary operator so that ≻ or ≽ can be used as in `a≻b` to compare `a` and `b`;

3. the use of generic types so that the same operator can be used for real valued numbers, integers, or combinations of these; and,

4. the use of `export` to make the operator available without needing to qualify it with the package name.

For multi-objective problems, the value z returned by the objective function will be a vector of values. When comparing multi-objective solutions, the comparison is based on dominance: a solution dominates another solution if the first one is at least as good in each individual value and strictly better for at least one value:

```
dominates(a, b) = all(a .≽ b) && any(a .≻ b)
```

This illustrates the use of *broadcast*, the dot operator, which asks Julia to apply the operator (or function) that follows the dot to each individual element in turn. So this code says that a dominates b if all the values of a are at least as good as the corresponding values of b and if any value of a is better than the corresponding value of b.

Given the definition of dominance, we can now define an operator for comparison:

```
≻(a :: Vector{T}, b :: Vector{T}) where T <: Number = dominates(a,b)
```

The important feature of this code is that we have defined the function to works for vectors so long as

1. they are of the same type, `T`, and
2. the type `T` represents a number entity, such as `Float64` or `Int32`.

Julia has a hierarchy of types defined.

The multiple dispatch feature of Julia will ensure that the proper comparison function is invoked when comparing objective function values for different points in the search.

Multiprocessing

An additional capability of Julia is multiprocessing, using multiple computers or computer cores simultaneously. Given the increasing prevalence of multi-core systems, including both desktop computers and laptops, it is reasonable to consider writing all code to make use of the extra potential computational power available.

In Julia, the `Threads` package which is part of the base system provides an easy to use interface to enable the use of multiple cores. The simplest construct, which I will be using in this book, is:

```
Threads.@threads for x ∈ acollection
    # do something(x)
end
```

This executes the body of the `for` loop in parallel using as many threads as made available to Julia. Invoking Julia with the `-t` option tells Julia how many cores to use simultaneously:

```
julia -t N
```

where `N` is some number, usually less than or equal to the number of cores available on the computer, or by

```
julia -t auto
```

In this case, Julia itself will determine the number of cores to use automatically.

The advantage of the nature inspired optimization methods I will be presenting later in this book is that they are all based on populations of solutions. Therefore, the computation associated with the members of the population are easy to distribute amounts the computational cores available. The plant propagation algorithm implementation (see Section 2.5) uses threads to evaluate members of the new population in parallel.

1.6 Representation

Different mathematical problem *formulations* of the same problem may have an impact on the solution process. Examples exist for the Simplex method [Hall and McKinnon 2004]. The choice of formulation may be a consideration for some problems. Further, for a given formulation, there may be alternative representations of the decision variables [Fraga et al. 2018] and these may affect the performance of individual optimization methods. Tailoring the formulation or representation to the method used may prove beneficial [Salhi and Vazquez-Rodriguez 2014].

For a given formulation, the representation should be chosen taking into account the following considerations:

1. The representation should be suitable for the optimization method or methods, enabling the implementation of the appropriate operations that are required by the methods. For instance, if the method is a genetic algorithm, the representation should be suitable for crossover and mutation operators.

2. The representation should cover the complete space of feasible solutions to the problem or, at worst, ensure that the desirable potential solutions can be represented.

3. On the other hand, the representation should minimise the probability of the method's operators generating infeasible or undesirable solutions.

Multiple dispatch in Julia aids the process of considering different representations. For instance, different versions of the objective function implementation can be written that differ in the *type* of the first argument. This enables alternative representations without needing to change the implementation of the solution method. An example of this appeared recently [Fraga 2021c]. This latter example includes code in Julia using the Fresa[15] implementation of a plant propagation algorithm [Fraga 2021b; Fraga 2021a] (see Section 2.5).

[15] `http://github.com/ericsfraga/Fresa.jl`

Bibliography

Ab Aziz, N.A., Alias, M.Y., Mohemmed, A.W., and Ab Aziz, K. 2011. Particle swarm optimization for constrained and multiobjective problems: A brief review. *Management and artificial intelligence*, Int Assoc Computer Science & Information Technology Press-IACSIT Press, 146+.

Bates, R.A., Wynn, H.P., and Fraga, E.S. 2007. Feasible region approximation: a comparison of search cone and convex hull methods. *Engineering optimization 39*, 5, 513–527. doi:10.1080/03052150701351680.

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

Biegler, L., Grossmann, I., and Westerberg, A. 1997. *Systematic methods of chemical process design*. Prentice-Hall.

Boston, J. and Britt, H. 1978. A radically different formulation and solution of the single-stage flash problem. *Computers & chemical engineering 2*, 2, 109–122. doi:10.1016/0098-1354(78)80015-5.

Burre, J., Bongartz, D., and Mitsos, A. 2022. Comparison of minlp formulations for global superstructure optimization. *Optimization and engineering*. doi:10.1007/s11081-021-09707-y.

Chen, J.J. 1987. Comments on improvements on a replacement for the logarithmic mean. *Chemical Engineering Science 42*, 10, 2488–2489.

Coello, C., Pulido, G., and Lechuga, M. 2004. Handling multiple objectives with particle swarm optimization. *IEEE transactions on evolution-*

ary computation 8, 3, 256–279. doi:10.1109/tevc.2004.826067.

Cui, Y., Meng, X., and Qiao, J. 2022. A multi-objective particle swarm optimization algorithm based on two-archive mechanism. *Applied soft computing 119*, 108532. doi:10.1016/j.asoc.2022.108532.

Dantzig, G.B. 1982. Reminiscences about the origins of linear programming. *Operations research letters 1*, 2, 43–48. doi:10.1016/0167-6377(82)90043-8.

Datta, D. and Figueira, J.R. 2011. A real-integer-discrete-coded particle swarm optimization for design problems. *Applied soft computing 11*, 4, 3625–3633. doi:10.1016/j.asoc.2011.01.034.

Deb, K. 2000. An efficient constraint handling method for genetic algorithms. *Comput. methods appl. mech. engng. 186*, 311–338.

Emmerich, M., Grötzner, M., and Schütz, M. 2001. Design of graph-based evolutionary algorithms: A case study for chemical process networks. *Evolutionary computation 9*, 3, 329–354. doi:10.1162/106365601750406028.

Feng, Q., Li, Q., Chen, P., et al. 2019. Multiobjective particle swarm optimization algorithm based on adaptive angle division. *IEEE access 7*, 87916–87930. doi:10.1109/ACCESS.2019.2925540.

Floudas, C.A. 1995. *Nonlinear and mixed-integer optimization. fundamentals and applications*. Oxford University Press, Oxford.

Floudas, C.A., Ciric, A.R., and Grossmann, I.E. 1986. Automatic synthesis of optimum heat-exchanger network configurations. *AIChE journal 32*, 2, 276–290.

Fraga, E., Patel, R., and Rowe, G. 2001a. A visual representation of process heat exchange as a basis for user interaction and stochastic optimization. *Chemical engineering research and design 79*, 7, 765–776. doi:10.1205/026387601753191948.

Fraga, E., Patel, R., and Rowe, G. 2001b. A visual representation of process heat exchange as a basis for user interaction and stochastic optimization. *Chemical engineering research and design 79*, 7, 765–776. doi:10.1205/026387601753191948.

Fraga, E.S. 1996. The automated synthesis of complex reaction/separation processes using dynamic programming. *Chemical engineering research* *and design 74*, A, 249–260.

Fraga, E.S. 2006. Hybrid methods for optimisation. In: J. Zilinskas and I.D.L. Bogle, eds., *Computer aided methods for optimal design and operations*. World Scientific Publishing Co., 1–14.

Fraga, E.S. 2008. A Lindenmayer system for heat exchanger network design with stream splitting. *Adaptive comuting in design and manufacture 2008: Proceedings of the eighth international conference*.

Fraga, E.S. 2009. A rewriting grammar for heat exchanger network structure evolution with stream splitting. *Engineering optimization 41*, 9, 813–831. doi:10.1080/03052150903070153.

Fraga, E.S. 2021a. ericsfraga/Fresa.jl: First public release (r2021.06.30). doi:10.5281/zenodo.5045812 doi:10.5281/zenodo.5045812.

Fraga, E.S. 2021b. Fresa: A plant propagation algorithm for black-box single and multiple objective optimization. *International journal on engineering technologies and informatics 2*, 4. doi:10.51626/i-jeti.2021.02.00022.

Fraga, E.S. 2021c. Multiple simultaneous solution representations in a population based evolutionary algorithm. arXiv:2106.05096Retrieved from `https://arxiv.org/abs/2106.05096`.

Fraga, E.S. and Amusat, O. 2016. Understanding the impact of constraints: a rank based fitness function for evolutionary methods. In: P.M. Pardalos, A. Zhigljavsky and J. Zilinskas, eds., *Advances in stochastic and deterministic global optimization*. Springer, 243–254. doi:10.1007/978-3-319-29975-4.

Fraga, E.S. and McKinnon, K.I.M. 1994. The use of dynamic programming with parallel computers for process synthesis. *Computers & chemical engineering 18*, 1, 1–13.

Fraga, E.S., Salhi, A., and Talbi, E.-G. 2018. On the impact of representation and algorithm selection for optimisation in process design: motivating a meta-heuristic framework. In: L. Amodeo, E.-G. Talbi and F. Yalaoui, eds., *Recent developments in metaheuristics*. Springer, 141–147.

Fraga, E.S., Steffens, M.A., Bogle, I.D.L., and Hind, A.K. 2000. An object oriented framework for process synthesis and simulation. *Foun-*

dations of computer-aided process design, 446–449.

Goh, K., Lim, A., and Rodrigues, B. 2003. Sexual selection for genetic algorithms. *Artificial intelligence review 19*, 2, 123–152. doi:10.1023/A:1022692631328.

Goldberg, D.E. 1989. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

Grossmann, I.E. and Kravanja, Z. 1995. Mixed-integer nonlinear programming techniques for process systems engineering. *Computers & chemical engineering 19*, Suppl., S189–S204.

Haimes, Y. 1971. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics SMC-1*, 3, 296–297. doi:10.1109/tsmc.1971.4308298.

Hall, J.A.J. and McKinnon, K.I.M. 2004. The simplex examples where the simplex method cycles and conditions where expand fails to prevent cycling. *Mathematical programming 100*, 133–150.

Holland, J.H. 1975. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI.

Katoch, S., Chauhan, S.S., and Kumar, V. 2021. A review on genetic algorithm: past, present, and future. *Multimedia tools and applications 80*, 5, 8091–8126. doi:10.1007/s11042-020-10139-6.

Kay, A.C. 1993. The early history of Smalltalk. Retrieved from `http://worrydream.com/EarlyHistoryOfSmalltalk/`.

Kelley, C.T. 1999. *Iterative methods for optimization*. SIAM.

Kennedy, J. and Eberhart, R. 1995. Particle swarm optimization. *Proceedings of ICNN'95 - International conference on neural networks*, 1942–1948. doi:10.1109/ICNN.1995.488968.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. 1983. Optimization by simulated annealing. *Science 220*, 4598, 671–680.

Knuth, D.E. 1984. Literate programming. *The computer journal 27*, 2, 97–111. doi:10.1093/comjnl/27.2.97.

Kondili, E., Pantelides, C., and Sargent, R. 1993. A general algorithm for short-term scheduling of batch operations—I. MILP formulation.

Computers & chemical engineering 17, 2, 211–227. doi:10.1016/0098-1354(93)80015-F.

Lara-Montaño, O.D., Gómez-Castro, F.I., and Gutierrez-Antonio, C. 2022. Development of a virtual platform for the metaheuristic optimization of heat exchangers. *Proceedings of the 32nd European Symposium on Computer Aided Process Engineering (ESCAPE32)*. doi:10.1016/B978-0-323-95879-0.50200-9.

Linhoff, B. and Flower, J.R. 1978. Synthesis of heat exchange networks, I & II. *AIChE journal 24*, 633.

Linnhoff, B. and Hindmarsh, E. 1983. The pinch design method for heat-exchanger networks. *Chemical Engineering Science 38*, 5, 745–763.

Marsili Libelli, S. and Alba, P. 2000. Adaptive mutation in genetic algorithms. *Soft computing 4*, 2, 76–80. doi:10.1007/s005000000042.

Monroy-Loperena, R. and Vacahern, M. 2012. Roots of the underwood's equations in short-cut distillation from a companion matrix eigenvalues. *Chemical engineering science 76*, 9–13. doi:10.1016/j.ces.2012.03.025.

Morton, W. 2002. Optimisation of a heat exchanger network superstructure using nonlinear programming. *Proc. inst. mech. eng. part e 216*, 2, 89–104.

Niu, Y. and Shen, L. 2007. Multiobjective constriction particle swarm optimization and its performance evaluation. *Advanced intelligent computing theories and applications, proceedings: With aspects of artificial intelligence*, IEEE Computat Intelligence Soc; Int Neural Network Soc; Natl Sci Fdn China; SPRINGER-VERLAG BERLIN, 1131–1140.

Ponton, J. 1989. Analytical approximations for the fixed-temperature phase split calculation. *Chemical engineering science 44*, 3, 769–773. doi:10.1016/0009-2509(89)85051-1.

Rackauckas, C. and Nie, Q. 2017. Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of open research software 5*, 1.

Raquel, C.R. and Naval, P.C. 2005. An effective use of crowding distance in multiobjective particle swarm optimization. *Proceedings of the 7th annual conference on genetic and evolution-*

ary computation, Association for Computing Machinery, 257–264. doi:10.1145/1068009.1068047.

RATHORE, R.N.S., VAN WORMER, K.A., AND POWERS, G.J. 1974. Synthesis strategies for multicomponent separation systems with energy integration. *AIChE j. 20*, 3, 491–502.

RIQUELME, N., VON LÜCKEN, C., AND BARAN, B. 2015. Performance metrics in multi-objective optimization. *2015 latin american computing conference (clei)*, 1–11. doi:10.1109/CLEI.2015.7360024.

RODMAN, A.D., FRAGA, E.S., AND GEROGIORGIS, D. 2018. On the application of a nature-inspired stochastic evolutionary algorithm to constrained multi-objective beer fermentation optimisation. *Computers & chemical engineering 108*, 448–459. doi:10.1016/j.compchemeng.2017.10.019.

SALHI, A. AND FRAGA, E.S. 2011. Nature-inspired optimisation approaches and the new plant propagation algorithm. *Proceedings of ICeMATH 2011, the international conference on numerical analysis and optimization*, K2:1–8.

SALHI, A. AND VAZQUEZ-RODRIGUEZ, J.A. 2014. Tailoring hyper-heuristics to specific instances of a scheduling problem using affinity and competence functions. *Journal of memetic computing*. doi:10.1007/s12293-013-012107.

SCHULTE, E. AND DAVISON, D. 2011. Active documents with org-mode. *Computing in science engineering 13*, 3, 66–73. doi:10.1109/MCSE.2011.41.

SHITAHUN, A., RUGE, V., GEBREMEDHIN, M., ET AL. 2013. Model-based dynamic optimization with openmodelica and casadi. *IFAC Proceedings Volumes 46*, 21, 446–451. doi:10.3182/20130904-4-JP-2042.00166.

SHU, X., LIU, Y., ZHANG, Q., AND YANG, M. 2022. A novel multiobjective particle swarm optimization combining hypercube and distance. *Scientific programming 2022*, 9448419. doi:10.1155/2022/9448419.

TOWLER, G. AND SINNOTT, R. 2013. *Chemical engineering design: Principles, practice and economics of plant and process design*. Butterworth-Heinemann.

VASSILIADIS, V.S., KÄHM, W., DEL RIO CHANONA, E.A., AND YUAN, Y. 2021. *Optimization for chemical and biochemical engineering: Theory,* *algorithms, modeling and applications*. Cambridge University Press. doi:10.1017/9781316227268.

Vrielink, W. and van den Berg, D. 2021a. A dynamic parameter for the plant propagation algorithm. *Evo* 2021*, 5.

Vrielink, W. and van den Berg, D. 2021b. Parameter control for the plant propagation algorithm. *Evo* 2021*, 1.

Zamora, J.M. and Grossmann, I.E. 1998. A global MINLP optimization algorithm for the synthesis of heat exchanger networks with no stream splits. *Computers & chemical engineering 22*, 3, 367–384.

Zhang, Q., Liu, Y., Han, H., Yang, M., and Shu, X. 2022. Multi-objective particle swarm optimization with multi-archiving strategy. *Scientific programming 2022*, 7372450. doi:10.1155/2022/7372450.

Zheng, Y.-J. and Chen, S.-Y. 2013. Cooperative particle swarm optimization for multiobjective transportation planning. *Applied intelligence 39*, 1, 202–216. doi:10.1007/s10489-012-0405-5.

Zilinskas, A., Fraga, E.S., and Mackutė, A. 2006. Data analysis and visualisation for robust multi-criteria process optimisation. *Computers & chemical engineering 30*, 6-7, 1061–1071.

Index