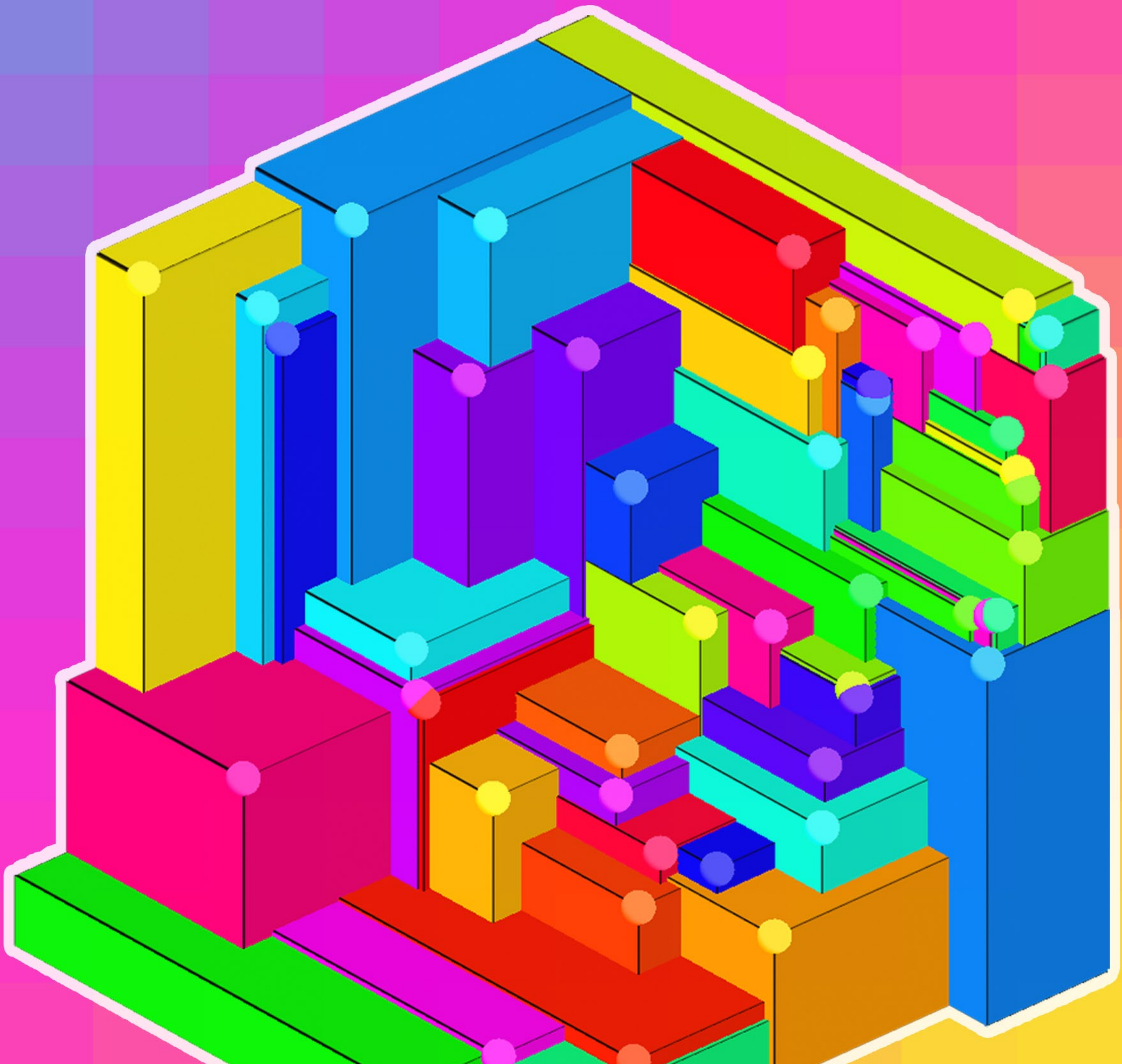
PRACTICAL EVOLUTIONARY ALGORITHMS

DR. SHAHIN ROSTAMI



Biography

Dr. Shahin Rostami is a Senior Academic (Associate Professor) and Principal Consultant in Data Science, Artificial Intelligence, Digital Health, and Defence Systems. He is currently working with industry whilst on his sabbatical from his position at the Department of Computing and Informatics at the Bournemouth University, where he has been a faculty member since 2014.

Dr. Rostami holds a Ph.D. in the field of Computational Intelligence with applications to Concealed Weapon Detection. His research interests lie within Data Science and Artificial Intelligence, ranging from theory to their application to Digital Healthcare and Threat Detection. Currently, he is consulting for and supervising PhD research projects in Non-Contact Vital Sign Measurement and Multi-Objective Concealed Weapon Detection. He has published in many high-impact journals and conferences, and organised/chaired special sessions including the IEEE CIBCB 2017 "Machine Learning in Medical Diagnosis and Prognosis". He also leads the Computational Intelligence Research Initiative (CIRI), and currently supervises 5 Ph.D. and 3 Ms.c. students in related subjects.

Dr. Rostami has founded and held the position of Programme Leader for multiple programmes at the postgraduate level: MS.c. Data Science and Artificial Intelligence (DSAI); MS.c. Digital Health and Artificial Intelligence (DHAI); and MS.c. Applied Data Analytics (ADA). He has designed and taught both postgraduate and undergraduate curriculum, such as Search and Optimisation, Artificial Intelligence, and Data Mining and Analytic Technologies. He is also committed to teaching Data Science and Computer Science to anyone, regardless of their educational background, e.g. his public videos on YouTube.

shahinrostami.com
YT: ShahinRostami
@ShahinRostami
Github: shahinrostami
u/shahinrostami
Patreon: StamiLabs.com
IG: Data.Crayon

Table of Contents

Preface

Preface

Evolutionary Algorithms (*EAs*) are a fascinating class of algorithms for meta-heuristic optimisation. There exist many books on the topic of EAs, ranging from their theory to practice. The first book I read on the topic was *Genetic algorithms in search, optimization, and machine learning* by David E. Goldberg (1989), and to this day I still recommend it to new students in the field.

However, there is a re-occurring difficulty when my students are starting out in the field, *"how do I move from theory to practice?"*. Most books will have some chapters dedicated to applications of EAs, but what's missing is an up-to-date book dedicated to using modern technology and concepts.

When writing this book, I had to answer some difficult questions:

- What programming language will my examples be written in?
- What software libraries will I use?
- How do I structure the chapters and sections, do I lead entirely by example or do I dedicate some parts to the theory?
- Do I focus on single-objective EAs or multi-objective EAs?

Nevertheless, the decisions had to be made. I selected Python as the programming language simply due to its rise in popularity (in 2019), and this was only a difficult choice because there is a wealth of resources written for MATLAB. Of the resources written in MATLAB, it is a shame to not be able to use PlatEMO, which is a well-maintained open-source platform for Evolutionary Multi-objective Optimisation. In its place, when a software library is needed, I will turn to Platypus, which provides optimisation algorithms and analysis tools for multi-objective optimisation.

For the structure of the chapters and sections, I have decided to lead entirely by example. There will be code to demonstrate every concept used, and I will show how we can implement algorithms from their mathematical representation. In these cases, I will focus on the readability of the implementations rather than their performance.

Finally, I will focus on multi-objective EAs as this represents the majority of real-world problems. However, single-objective EAs will make an appearance to highlight the differences between the two.

Perhaps the most difficult question to answer is *where do we start?* There is so much to cover, and many potential starting points. For this book, I will start with a definition of objective functions, and illustrate the relationship between what we call the problem space and the objective space. With this approach, I hope there will be a clear understanding of what the various operators within an EA are affecting.

Note

I aim to generate everything in this book through code. This means you will see the code for all my figures and tables, including things like flowcharts.

Every section is intended to be independent, so you will find some repetition as you progress from one section to another.

Synthetic Objective Functions and ZDT1

Contents Download Source

Preamble

```
# used to create block diagrams
%reload_ext xdiag_magic
%xdiag_output_format svg

import numpy as np                    # for multi-dimensional containers
import pandas as pd                   # for DataFrames
import plotly.graph_objects as go     # for data visualisation
```

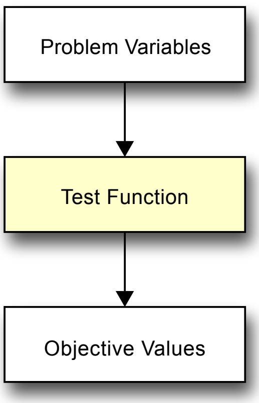
Introduction

In mathematics, optimisation is concerned with the selection of optimal solutions to objective functions. An objective function consists of input arguments referred to as problem variables (or genotype) which are computed by one or many mathematical functions to determine the objective value (or phenotype).

Real-world optimisation problems are divided into one (in the case of single-objective optimisation) or many (in the case of multi-objective optimisation) objective functions in order to be optimised by an optimisation algorithm. The difficulty of convergence can be reduced by the bounding of problem variables as this reduces the size of the search domain.

In order to determine an Evolutionary Algorithm's robustness when solving problems consisting of multiple objectives, its performance must be assessed on the optimisation of synthetic test functions which are created for the purpose of testing. These problems may also be used to systematically compare two or more Evolutionary Algorithms.

```
%%blockdiag
{
    orientation = portrait
    "Problem Variables" -> "Test Function" -> "Objective Values"
    "Test Function" [color = '#ffffcc']
}
```



Synthetic test functions are typically:

- **Intentionally difficult**, meaning they are designed to include optimisation difficulties which are present in real-world problems.
- **Scalable**, meaning they can be configured with a different number of problem variables and objectives.
- **Computationally efficient**, meaning they are faster to execute than a real-world problem. This is desirable when benchmarking an Evolutionary Algorithm.

In contrast, real-world problems which have been encapsulated within an objective function in order to be used by an optimiser are often computationally expensive and have long execution times. This is because synthetic test functions are often mathematical equations which aim to cause difficulty for an optimiser when searching for problem variables that produce optimal objective values, whereas real-world problems often involve computationally expensive simulations in order to arrive at the objective values.

Put simply, using a real-world problem to evaluate the performance of a newly proposed Evolutionary Algorithm only allows us to determine if an algorithm is good at solving that single problem. What we're interested in is analysing how Evolutionary Algorithms perform when encountering various difficulties that appear in multi-objective problems, and how they compare to each other.

The ZDT1 test function

We will be using a synthetic test problem throughout this notebook called ZDT1. It is part of the ZDT test suite, consisting of six different two-objective synthetic test problems. This is quite an old test suite, easy to solve, and very easy to visualise.

Mathematically, the ZDT1[1] two-objective test function can be expressed as:

$$
\begin{aligned}
f_1(x_1) &= x_1 \\
f_2(x) &= g \cdot h \\
g(x_2, \ldots, x_{\mathrm{D}}) &= 1 + 9 \cdot \sum_{d=2}^{\mathrm{D}} \frac{x_d}{(D-1)} \\
h(f_1, g) &= 1 - \sqrt{f1/g}
\end{aligned} \tag{1}
$$

where x is a solution to the problem, defined as a vector of D decision variables.

$$x = \langle x_1, x_2, \ldots, x_{\mathrm{D}} \rangle \tag{2}$$

and all decision variables fall between 0 and 1.

$$0 \leq x_d \leq 1, d = 1, \ldots, \mathrm{D} \tag{3}$$

For this bi-objective test function, f_1 is the first objective, and f_2 is the second objective. This particular objective function is, by design, scalable up to any number of problem variables but is restricted to two problem objectives.

Let's start implementing this in Python, beginning with the initialisation of a solution according to Equations 2 and 3. In this case, we will have 30 problem variables $\mathrm{D} = 30$.

```python
D = 30
x = np.random.rand(D)
print(x)
```

```
[0.9204285  0.94911763 0.58171075 0.86276293 0.74091143 0.76786472
 0.32042083 0.99105863 0.82828293 0.41936616 0.43302201 0.3502152
 0.52507767 0.02871106 0.21002611 0.14115373 0.54289036 0.0767823
 0.67900898 0.54012128 0.70848884 0.92282082 0.56027323 0.93871461
 0.06385584 0.597213   0.53146707 0.27110393 0.95440343 0.01783687]
```

Now that we have a solution to evaluate, let's implement the ZDT1 synthetic test function using Equation 1.

```python
def ZDT1(x):
    f1 = x[0]  # objective 1
    g = 1 + 9 * np.sum(x[1:D] / (D-1))
    h = 1 - np.sqrt(f1 / g)
    f2 = g * h  # objective 2

    return [f1, f2]
```

Finally, let's invoke our implemented test function using our solution x from earlier.

```
objective_values = ZDT1(x)
print(objective_values)
```

```
[0.9204284983552486, 3.5113646524980835]
```

Now we can see the two objective values that measure our solution x according to the ZDT1 synthetic test function, which is a minimisation problem.

Performance in Objective Space

We will be discussing desirable characteristics in multi-objective solutions, but for now, let's plot some randomly initialised solutions against an optimal set of solutions for ZDT1. This is a synthetic test function, and as such the authors have provided us with a way to calculate the optimal set.

$$f_2 = 1 - \sqrt{f_1} \tag{4}$$

Let's use this to generate 20 ideal sets of objective values.

```
true_front = np.empty((0, 2))

for f1 in np.linspace(0, 1, num=20):
    f2 = 1 - np.sqrt(f1)
    true_front = np.vstack([true_front, [f1, f2]])

true_front = pd.DataFrame(true_front, columns=['f1', 'f2'])  # convert to
DataFrame
true_front
```

	f1	f2
0	0.000000	1.000000
1	0.052632	0.770584
2	0.105263	0.675557
3	0.157895	0.602640
4	0.210526	0.541169
5	0.263158	0.487011
6	0.315789	0.438049
7	0.368421	0.393023
8	0.421053	0.351114
9	0.473684	0.311753
10	0.526316	0.274524
11	0.578947	0.239114

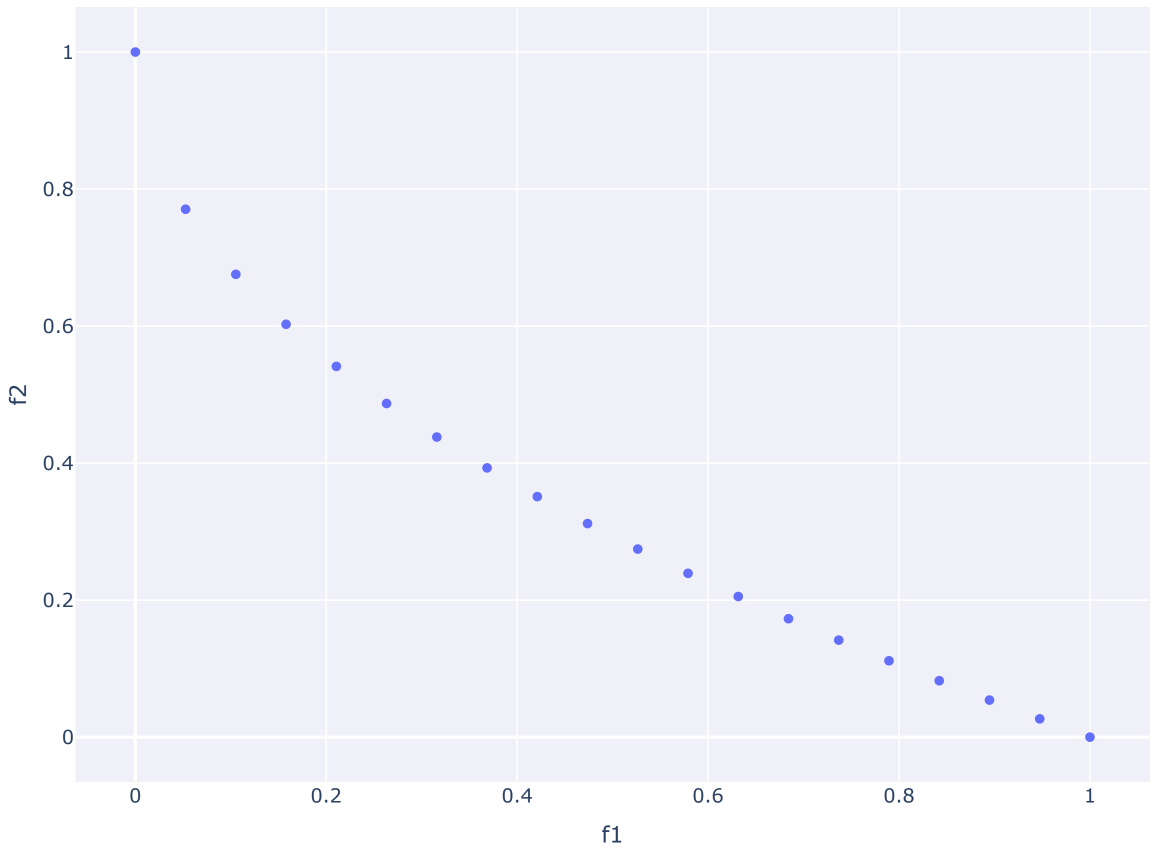
12	0.631579	0.205281
13	0.684211	0.172830
14	0.736842	0.141605
15	0.789474	0.111477
16	0.842105	0.082337
17	0.894737	0.054095
18	0.947368	0.026671
19	1.000000	0.000000

Now we can plot them to have an idea of the shape of the true front for ZDT1 in objective space.

```
fig = go.Figure(layout=dict(xaxis=dict(title='f1'),yaxis=dict(title='f2')))

fig.add_scatter(x=true_front.f1, y=true_front.f2, mode='markers')

fig.show()
```



To wrap things up, let's generate 50 objective values using the ZDT1 objective function created above. We achieve this by passing in 50 randomly initialised sets of problem variables.

```python
objective_values = np.empty((0, 2))

for i in range(50):
    x = np.random.rand(D)
    y = ZDT1(x)
    objective_values = np.vstack([objective_values, y])

# convert to DataFrame
objective_values = pd.DataFrame(objective_values, columns=['f1', 'f2'])
objective_values.head()
```

	f1	f2
0	0.969711	2.900818
1	0.731999	2.945158
2	0.144458	3.793344
3	0.337195	4.329990
4	0.767939	3.841221

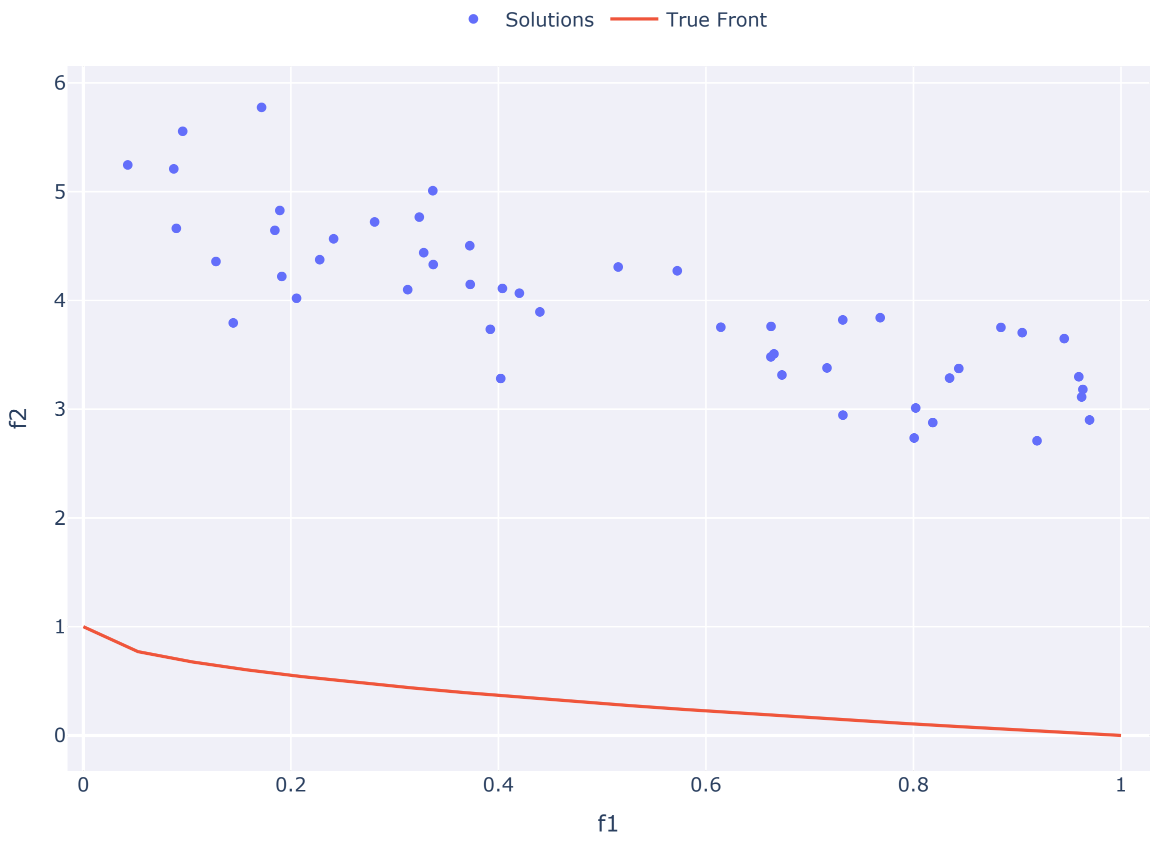
Now we will plot the objective values of our randomly initialised solutions on top of a plot of the true front, this will give us some idea of the difference in performance between the two sets.

```python
fig = go.Figure(layout=dict(xaxis=dict(title='f1'), yaxis=dict(title='f2')))

fig.add_scatter(x=objective_values.f1, y=objective_values.f2,
                name='Solutions', mode='markers')

fig.add_scatter(x=true_front.f1, y=true_front.f2, name='True Front')

fig.show()
```



Conclusion

In this section, we introduced the concept of synthetic test functions along with ZDT1, a popular and relatively easy example. We expressed the concept mathematically and then made a direct implementation using Python. We then generated a set of 50 solutions, calculated the objective values for each one, and plotted the objective space using a scatter plot.

There are many suites of synthetic test functions in the literature, some to read about are ZDT, DTLZ, CEC09, and WFG Toolkit.

Exercise

Using the ZDT test suite paper listed in the references, duplicate this notebook but with the focus on ZDT2 instead of ZDT1.

---

1. E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Evolutionary Computation, 8(2):173-195, 2000 ↩