合肥大學
HEFEI UNIVERSITY

Metaheuristic Optimization in Python: moptipy

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2026-02-20

Outline

Introduction

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].
- Its core website is `https://thomasweise.github.io/moptipy`.

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].
- Its core website is `https://thomasweise.github.io/moptipy`.
- moptipy offers different optimization algorithms, experiment execution, and result evaluation facilities.

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].
- Its core website is `https://thomasweise.github.io/moptipy`.
- moptipy offers different optimization algorithms, experiment execution, and result evaluation facilities.
- It allows you to execute fully replicable and self-documenting experiments in a parallel and distributed fashion.

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].
- Its core website is `https://thomasweise.github.io/moptipy`.
- moptipy offers different optimization algorithms, experiment execution, and result evaluation facilities.
- It allows you to execute fully replicable and self-documenting experiments in a parallel and distributed fashion.
- Moreover, it also provides the tools to statistically evaluate the experimental results and to plot various diagrams and charts.

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].
- Its core website is `https://thomasweise.github.io/moptipy`.
- moptipy offers different optimization algorithms, experiment execution, and result evaluation facilities.
- It allows you to execute fully replicable and self-documenting experiments in a parallel and distributed fashion.
- Moreover, it also provides the tools to statistically evaluate the experimental results and to plot various diagrams and charts.
- Additionally, it ships with two example application areas that can be used for testing algorithms, namely the Job Shop Scheduling Problem (JSSP)[65,68,70].

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].
- Its core website is `https://thomasweise.github.io/moptipy`.
- moptipy offers different optimization algorithms, experiment execution, and result evaluation facilities.
- It allows you to execute fully replicable and self-documenting experiments in a parallel and distributed fashion.
- Moreover, it also provides the tools to statistically evaluate the experimental results and to plot various diagrams and charts.
- Additionally, it ships with two example application areas that can be used for testing algorithms, namely the Job Shop Scheduling Problem (JSSP)[65,68,70] and discrete optimization benchmarks such as those used in[70,71].

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].
- Its core website is https://thomasweise.github.io/moptipy.
- moptipy offers different optimization algorithms, experiment execution, and result evaluation facilities.
- It allows you to execute fully replicable and self-documenting experiments in a parallel and distributed fashion.
- Moreover, it also provides the tools to statistically evaluate the experimental results and to plot various diagrams and charts.
- Additionally, it ships with two example application areas that can be used for testing algorithms, namely the Job Shop Scheduling Problem (JSSP)[65,68,70] and discrete optimization benchmarks such as those used in[70,71].
- Its companion package moptipyapps (https://thomasweise.github.io/moptipyapps) offers even more example application areas.

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].
- Its core website is `https://thomasweise.github.io/moptipy`.
- moptipy offers different optimization algorithms, experiment execution, and result evaluation facilities.
- It allows you to execute fully replicable and self-documenting experiments in a parallel and distributed fashion.
- Moreover, it also provides the tools to statistically evaluate the experimental results and to plot various diagrams and charts.
- Additionally, it ships with two example application areas that can be used for testing algorithms, namely the Job Shop Scheduling Problem (JSSP)[65,68,70] and discrete optimization benchmarks such as those used in[70,71].
- Its companion package moptipyapps (`https://thomasweise.github.io/moptipyapps`) offers even more example application areas, including the Quadratic Assignment Problem (QAP)[11,58].

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].
- Its core website is `https://thomasweise.github.io/moptipy`.
- moptipy offers different optimization algorithms, experiment execution, and result evaluation facilities.
- It allows you to execute fully replicable and self-documenting experiments in a parallel and distributed fashion.
- Moreover, it also provides the tools to statistically evaluate the experimental results and to plot various diagrams and charts.
- Additionally, it ships with two example application areas that can be used for testing algorithms, namely the Job Shop Scheduling Problem (JSSP)[65,68,70] and discrete optimization benchmarks such as those used in[70,71].
- Its companion package moptipyapps (`https://thomasweise.github.io/moptipyapps`) offers even more example application areas, including the Quadratic Assignment Problem (QAP)[11,58], the Traveling Salesperson Problem (TSP)[38–40].

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].
- Its core website is `https://thomasweise.github.io/moptipy`.
- moptipy offers different optimization algorithms, experiment execution, and result evaluation facilities.
- It allows you to execute fully replicable and self-documenting experiments in a parallel and distributed fashion.
- Moreover, it also provides the tools to statistically evaluate the experimental results and to plot various diagrams and charts.
- Additionally, it ships with two example application areas that can be used for testing algorithms, namely the Job Shop Scheduling Problem (JSSP)[65,68,70] and discrete optimization benchmarks such as those used in[70,71].
- Its companion package moptipyapps (`https://thomasweise.github.io/moptipyapps`) offers even more example application areas, including the Quadratic Assignment Problem (QAP)[11,58], the Traveling Salesperson Problem (TSP)[38–40], the two-dimensional bin packing task[76,77].

Introduction to `moptipy`

- moptipy[69] – *Metaheuristic Optimization in Python* – as the name suggests, is a library with implementations of metaheuristic optimization methods in Python[66].
- Its core website is `https://thomasweise.github.io/moptipy`.
- moptipy offers different optimization algorithms, experiment execution, and result evaluation facilities.
- It allows you to execute fully replicable and self-documenting experiments in a parallel and distributed fashion.
- Moreover, it also provides the tools to statistically evaluate the experimental results and to plot various diagrams and charts.
- Additionally, it ships with two example application areas that can be used for testing algorithms, namely the Job Shop Scheduling Problem (JSSP)[65,68,70] and discrete optimization benchmarks such as those used in[70,71].
- Its companion package moptipyapps (`https://thomasweise.github.io/moptipyapps`) offers even more example application areas, including the Quadratic Assignment Problem (QAP)[11,58], the Traveling Salesperson Problem (TSP)[38–40], the two-dimensional bin packing task[76,77], as well as the Traveling Tournament Problem (TTP)[73].

Wishlist

- When doing experiments with optimization algorithms, there are several basic features that we would like to have.

Wishlist

- When doing experiments with optimization algorithms, there are several basic features that we would like to have:
  1. We want experiments to be well-documented.

Wishlist

- When doing experiments with optimization algorithms, there are several basic features that we would like to have:
  1. We want experiments to be well-documented.
  2. We want experiments to be replicable.

Wishlist

- When doing experiments with optimization algorithms, there are several basic features that we would like to have:
  1. We want experiments to be well-documented.
  2. We want experiments to be replicable.
  3. We want experiments to run in parallel or even distribute them over several computers.

Wishlist

- When doing experiments with optimization algorithms, there are several basic features that we would like to have:
  1. We want experiments to be well-documented.
  2. We want experiments to be replicable.
  3. We want experiments to run in parallel or even distribute them over several computers.
- Log files store the information collected during experiments.

Wishlist

- When doing experiments with optimization algorithms, there are several basic features that we would like to have:
  1. We want experiments to be well-documented.
  2. We want experiments to be replicable.
  3. We want experiments to run in parallel or even distribute them over several computers.
- Log files store the information collected during experiments.
- Their structure lays the foundation for fulfilling the wishlist.

Wishlist

- When doing experiments with optimization algorithms, there are several basic features that we would like to have:
  1. We want experiments to be well-documented.
  2. We want experiments to be replicable.
  3. We want experiments to run in parallel or even distribute them over several computers.
- Log files store the information collected during experiments.
- Their structure lays the foundation for fulfilling the wishlist: A good structure can lead to a natural propensity for parallelism, distribution, and self-documentation, a bad structure can make them impossible.

Wishlist

- When doing experiments with optimization algorithms, there are several basic features that we would like to have:
  1. We want experiments to be well-documented.
  2. We want experiments to be replicable.
  3. We want experiments to run in parallel or even distribute them over several computers.
- Log files store the information collected during experiments.
- Their structure lays the foundation for fulfilling the wishlist: A good structure can lead to a natural propensity for parallelism, distribution, and self-documentation, a bad structure can make them impossible.
- We here discuss how such features can be achieved.

Log Files in moptipy

Log Files should automatically contain...

- Log files should automatically contain the following information.

Log Files should automatically contain...

- Log files should automatically contain the following information:
  1. all information about the algorithm and its configuration.

Log Files should automatically contain...

- Log files should automatically contain the following information:
  1. all information about the algorithm and its configuration,
  2. all information about the problem instance.

Log Files should automatically contain...

- Log files should automatically contain the following information:
  1. all information about the algorithm and its configuration,
  2. all information about the problem instance,
  3. the random seed of the corresponding run.

Log Files should automatically contain...

- Log files should automatically contain the following information:
  1. all information about the algorithm and its configuration,
  2. all information about the problem instance,
  3. the random seed of the corresponding run,
  4. the end result: not just in terms of objective value, but the actual element of the solution space; also the point in search space if search $\neq$ solution space.

Log Files should automatically contain...

- Log files should automatically contain the following information:
  1. all information about the algorithm and its configuration,
  2. all information about the problem instance,
  3. the random seed of the corresponding run,
  4. the end result: not just in terms of objective value, but the actual element of the solution space; also the point in search space if search $\neq$ solution space,
  5. the total consumed FEs and runtime.

Log Files should automatically contain...

- Log files should automatically contain the following information:
  1. all information about the algorithm and its configuration,
  2. all information about the problem instance,
  3. the random seed of the corresponding run,
  4. the end result: not just in terms of objective value, but the actual element of the solution space; also the point in search space if search $\neq$ solution space,
  5. the total consumed FEs and runtime,
  6. the termination criteria.

Log Files should automatically contain...

- Log files should automatically contain the following information:
  1. all information about the algorithm and its configuration,
  2. all information about the problem instance,
  3. the random seed of the corresponding run,
  4. the end result: not just in terms of objective value, but the actual element of the solution space; also the point in search space if search $\neq$ solution space,
  5. the total consumed FEs and runtime,
  6. the termination criteria,
  7. optionally: the progress of the algorithm over time.

Log Files should automatically contain...

- Log files should automatically contain the following information:
  1. all information about the algorithm and its configuration,
  2. all information about the problem instance,
  3. the random seed of the corresponding run,
  4. the end result: not just in terms of objective value, but the actual element of the solution space; also the point in search space if search $\neq$ solution space,
  5. the total consumed FEs and runtime,
  6. the termination criteria,
  7. optionally: the progress of the algorithm over time,
  8. the system configuration, including processor, memory, Operating System (OS), library versions...

Log Files should automatically contain...

- Log files should automatically contain the following information:
  1. all information about the algorithm and its configuration,
  2. all information about the problem instance,
  3. the random seed of the corresponding run,
  4. the end result: not just in terms of objective value, but the actual element of the solution space; also the point in search space if search $\neq$ solution space,
  5. the total consumed FEs and runtime,
  6. the termination criteria,
  7. optionally: the progress of the algorithm over time,
  8. the system configuration, including processor, memory, Operating System (OS), library versions..., and
  9. exceptions and the stack trace if something went wrong.

1 Log File for 1 Run

- For every single run of one algorithm on one problem instance, one log file should be generated which can describe this run completely.

1 Log File for 1 Run

- For every single run of one algorithm on one problem instance, one log file should be generated which can describe this run completely.
- Information should not be divided over multiple files (harder to understand + loss of files?).

1 Log File for 1 Run

- For every single run of one algorithm on one problem instance, one log file should be generated which can describe this run completely.
- Information should not be divided over multiple files (harder to understand + loss of files?).
- No more than one run should be stored in one file (What if experiments fails after first run? What about parallelism?)

1 Log File for 1 Run

- For every single run of one algorithm on one problem instance, one log file should be generated which can describe this run completely.
- Information should not be divided over multiple files (harder to understand + loss of files?).
- No more than one run should be stored in one file (What if experiments fails after first run? What about parallelism?)
- Information should be easy-to-read and easy-to-parse text with a clear and rigid and self-documenting structure.

1 Log File for 1 Run

- For every single run of one algorithm on one problem instance, one log file should be generated which can describe this run completely.
- Information should not be divided over multiple files (harder to understand + loss of files?).
- No more than one run should be stored in one file (What if experiments fails after first run? What about parallelism?)
- Information should be easy-to-read and easy-to-parse text with a clear and rigid and self-documenting structure.
- No complicated format like JavaScript Object Notation (JSON), YAML Ain't Markup Language™ (YAML), or Extensible Markup Language (XML)! (Makes parsing complex. Different people may use different subsets of their functionality.)

`moptipy` Log File Structure

```
BEGIN_PROGRESS
fes;timeMS;f
1;6;267
5;7;235
10;7;230
20;7;227
25;7;205
40;7;200
84;8;180
END_PROGRESS
```

Progress of the Algorithm over Time

`fes` ... consumed objective function evaluations
`timeMS` ... consumed runtime in milliseconds
`f` ... **objective value of the `fes`-th evaluated solution**

Here , only moves improving the best-so-far solution are logged.

The 10th solution that was evaluated had objective value 230. Its evaluation was completed 7ms after the begin of the run.

```
BEGIN_STATE
totalFEs: 84
totalTimeMillis: 8
bestF: 180
lastImprovementFE: 84
lastImprovementTimeMillis: 8
END_STATE
```

End State at the end of the optimization run

- `totalFEs`: total consumed FEs
- `totalTimeMillis`: total consumed runtime in milliseconds (ms)
- `bestF`: best objective value encountered
- `lastImprovementFE`: FE when the last improvement took place
- `lastImprovementTimeMillis`: ms when the last improvement took place

```
BEGIN_SETUP
p.name: LoggingProcessWithSearchSpace
p.class: moptipy.api._process_ss_log._ProcessSSLog
p.lowerBound: 180
p.upperBound: 482
p.maxTimeMillis: 120000
p.goalF: 180
p.randSeed: 6526669205530947346
p.randSeed(hex): 0x5a9363100a272f12
p.randGenType: numpy.random._generator.Generator
p.randBitGenType: numpy.random._pcg64.PCG64
a.name: rls_swap2
a.class: moptipy.algorithms.so.rls.RLS
a.op0.name: shuffle
a.op0.class: moptipy.operators.permutations.
    op0_shuffle.Op0Shuffle
a.op1.name: swap2
a.op1.class: moptipy.operators.permutations.op1_swap2.Op1Swap2
y.name: gantt_demo
y.class: moptipy.examples.jssp.gantt_space.GanttSpace
y.shape: (5, 4, 3)
y.dtype: h
y.inst.name: demo
y.inst.class: moptipy.examples.jssp.instance.Instance
y.inst.machines: 5
y.inst.jobs: 4
y.inst.makespanLowerBound: 180
y.inst.makespanUpperBound: 482
y.inst.dtype: b
f.name: makespan
f.class: moptipy.examples.jssp.makespan.Makespan
f.lowerBound: 180
f.upperBound: 482
x.name: perm4w5r
x.class: moptipy.spaces.permutations.Permutations
x.nvars: 20
x.dtype: b
x.min: 0
x.max: 3
x.repetitions: 5
g.name: operation_based_encoding
g.class: moptipy.examples.jssp.
    ob_encoding.OperationBasedEncoding
g.dtypeMachineIdx: b
g.dtypeJobIdx: b
g.dtypeJobTime: h
END_SETUP
```

Algorithm and Problem Instance Information

- maximum runtime, if specified
- goal objective value, if specified
- random seed (decimal)
- random seed (hexadecimal)
- algorithm name
- algorithm class
- nullary operator
- unary operator
- solution space setup
- objective function
- lower and/or upper bound of objective function, if any
- search space setup
- decoder from search to solution space

```
BEGIN_SYS_INFO
session.start: 2023-06-20 04:51:52.514777+00:00
session.node: home
session.procesId: 0x3d08
session.cpuAffinity: 4;5
session.ipAddress: 192.168.1.109
version.Pillow: 9.4.0
version.contourpy: 1.0.6
version.cycler: 0.11.0
version.fonttools: 4.38.0
version.joblib: 1.2.0
version.kiwisolver: 1.4.4
version.llvmlite: 0.40.0
version.matplotlib: 3.7.1
version.moptipy: 0.9.76
version.numba: 0.57.0
version.numpy: 1.24.3
version.packaging: 21.3
version.pdfo: 1.3.1
version.psutil: 5.9.5
version.pyparsing: 3.0.9
version.pythondateutil: 2.8.2
version.scikitlearn: 1.2.2
version.scipy: 1.10.1
version.six: 1.16.0
version.threadpoolctl: 3.1.0
hardware.machine: x86_64
hardware.nPhysicalCpus: 8
hardware.nLogicalCpus: 16
hardware.cpuMhz: (2200MHz..3700MHz)*16
hardware.byteOrder: little
hardware.cpu: AMD Ryzen 7 2700X Eight-Core Processor
hardware.memSize: 33595326464
python.version: 3.10.6 (main, May 29 2023, 11:10:38)
    [GCC 11.3.0]
python.implementation: CPython
os.name: Linux
os.release: 5.19.0-45-generic
os.version: 46~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Wed Jun 7
    15:06:04 UTC 20
END_SYS_INFO
```

Information about the System Running the Experiment

- versions of libaries used
- information about system hardware
- Python interpreter infos
- operating system information

```
BEGIN_RESULT_X
2;1;3;1;0;0;2;0;1;2;3;1;0;2;1;3;0;3;2;3
END_RESULT_X
```

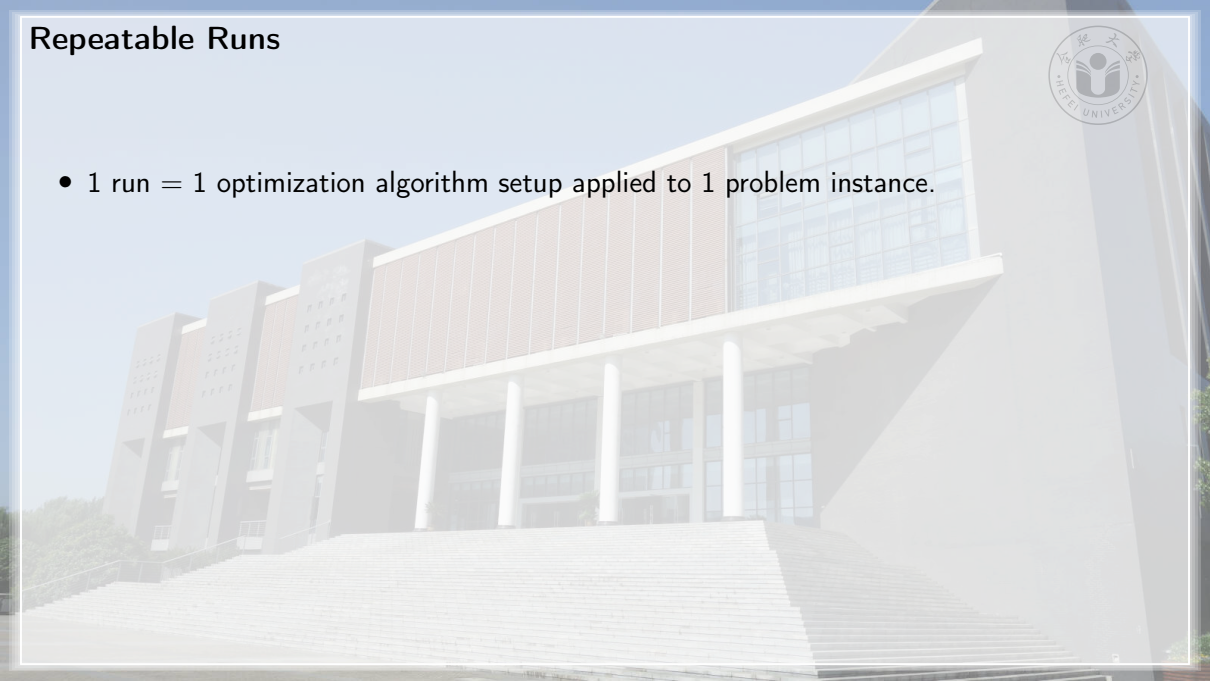
Best Encoded Solution (if any)

```
BEGIN_RESULT_Y
1;20;30;0;30;40;3;145;165;2;170;180;1;0;20;0;40;60;2;60;80;3;16
5;180;2;0;30;0;60;80;1;80;130;3;130;145;1;30;60;3;60;90;0;90;13
0;2;130;170;3;0;50;2;80;92;1;130;160;0;160;170
END_RESULT_Y
```

Best Discovered Point in the Solution Space

Repeatability and Replicability

Repeatable Runs



- 1 run = 1 optimization algorithm setup applied to 1 problem instance.

Repeatable Runs

- 1 run = 1 optimization algorithm setup applied to 1 problem instance.
- Repeatable runs = execute one run again and the algorithm will perform exactly the same operations at exactly the same time steps.

Repeatable Runs

- 1 run = 1 optimization algorithm setup applied to 1 problem instance.
- Repeatable runs = execute one run again and the algorithm will perform exactly the same operations at exactly the same time steps.
- . . . but randomized algorithms are . . . well . . . randomized?

Repeatable Runs

- 1 run = 1 optimization algorithm setup applied to 1 problem instance.
- Repeatable runs = execute one run again and the algorithm will perform exactly the same operations at exactly the same time steps.
- ... but randomized algorithms are ... well ... randomized?
- Use the same random seed $s \Longrightarrow$ algorithms become deterministic.

Repeatable Runs

- 1 run = 1 optimization algorithm setup applied to 1 problem instance.
- Repeatable runs = execute one run again and the algorithm will perform exactly the same operations at exactly the same time steps.
- . . . but randomized algorithms are . . . well . . . randomized?
- Use the same random seed $s \Longrightarrow$ algorithms become deterministic.
- Pass seeded random number generator to the algorithm.

Repeatable Runs

- 1 run = 1 optimization algorithm setup applied to 1 problem instance.
- Repeatable runs = execute one run again and the algorithm will perform exactly the same operations at exactly the same time steps.
- . . . but randomized algorithms are . . . well . . . randomized?
- Use the same random seed $s \implies$ algorithms become deterministic.
- Pass seeded random number generator to the algorithm.
- No other source of randomness allowed.

Repeatable Runs

- 1 run = 1 optimization algorithm setup applied to 1 problem instance.
- Repeatable runs = execute one run again and the algorithm will perform exactly the same operations at exactly the same time steps.
- ... but randomized algorithms are ... well ... randomized?
- Use the same random seed $s \implies$ algorithms become deterministic.
- Pass seeded random number generator to the algorithm.
- No other source of randomness allowed.
- The seeds s should be chosen fairly (no cherry picking possible) and in a well-defined, reproducible manner.

Seeds S for n Runs of Algorithm $\mathcal{A}$ on Instance $\mathcal{I}$

- $\mathrm{name}(\mathcal{I})$ be the user-defined name string of the problem instance $\mathcal{I}$.

Seeds S for n Runs of Algorithm $\mathcal{A}$ on Instance $\mathcal{I}$

- $\mathrm{name}(\mathcal{I})$ be the user-defined name string of the problem instance $\mathcal{I}$.
- It could be *"onemax_20"* for the 20-bit OneMax problem, or maybe *"swv15"* for an instance of the Job Shop Scheduling Problem (JSSP), for example.

Seeds S for n Runs of Algorithm $\mathcal{A}$ on Instance $\mathcal{I}$

- $\text{name}(\mathcal{I})$ be the user-defined name string of the problem instance $\mathcal{I}$.
- It could be *"onemax_20"* for the 20-bit OneMax problem, or maybe *"swv15"* for an instance of the Job Shop Scheduling Problem (JSSP), for example.
- Compute the SHA-512 digest[51] (hash) of the UTF-8 encoded[30,75] $\text{name}(\mathcal{I})$ and get 64 bytes of data D.

Seeds S for n Runs of Algorithm $\mathcal{A}$ on Instance $\mathcal{I}$

- $\mathrm{name}(\mathcal{I})$ be the user-defined name string of the problem instance $\mathcal{I}$.
- It could be *"onemax_20"* for the 20-bit OneMax problem, or maybe *"swv15"* for an instance of the Job Shop Scheduling Problem (JSSP), for example.
- Compute the SHA-512 digest[51] (hash) of the UTF-8 encoded[30,75] $\mathrm{name}(\mathcal{I})$ and get 64 bytes of data D.
- Divide the data D into two 32-byte chunks D_1 and D_2.

Seeds S for n Runs of Algorithm $\mathcal{A}$ on Instance $\mathcal{I}$

- $\text{name}(\mathcal{I})$ be the user-defined name string of the problem instance $\mathcal{I}$.
- It could be *"onemax_20"* for the 20-bit OneMax problem, or maybe *"swv15"* for an instance of the Job Shop Scheduling Problem (JSSP), for example.
- Compute the SHA-512 digest[51] (hash) of the UTF-8 encoded[30,75] $\text{name}(\mathcal{I})$ and get 64 bytes of data D.
- Divide the data D into two 32-byte chunks D_1 and D_2.
- Seed two of NumPy's PCG64 pseudo random number generators[45,46], $\mathfrak{R}_1$ and $\mathfrak{R}_2$, with D_1 and D_2, respectively.

Seeds S for n Runs of Algorithm $\mathcal{A}$ on Instance $\mathcal{I}$

- $\mathrm{name}(\mathcal{I})$ be the user-defined name string of the problem instance $\mathcal{I}$.
- It could be *"onemax_20"* for the 20-bit OneMax problem, or maybe *"swv15"* for an instance of the Job Shop Scheduling Problem (JSSP), for example.
- Compute the SHA-512 digest[51] (hash) of the UTF-8 encoded[30,75] $\mathrm{name}(\mathcal{I})$ and get 64 bytes of data D.
- Divide the data D into two 32-byte chunks D_1 and D_2.
- Seed two of NumPy's PCG64 pseudo random number generators[45,46], $\mathfrak{R}_1$ and $\mathfrak{R}_2$, with D_1 and D_2, respectively.
- Draw unsigned 64-bit integers as seeds s alternatingly from $\mathfrak{R}_1$ and $\mathfrak{R}_2$ until n unique values have been collected.

Advantages of this Method

- Random seeds are generated by a deterministic and replicable procedure.

Advantages of this Method

- Random seeds are generated by a deterministic and replicable procedure:
  - No cherry picking is possible.

Advantages of this Method

- Random seeds are generated by a deterministic and replicable procedure:
  - No cherry picking is possible.
  - If we first generate the set S_1 with $|S_1| = n_1$ seeds and then generate the set S_2 with $|S_2| = n_2$ seeds with $n_2 > n_1$ for the same problem instance $\mathcal{I}$, then $S_1 \subset S_2$, i.e., we can iteratively increase the number of runs in an experiment.

Advantages of this Method

- Random seeds are generated by a deterministic and replicable procedure:
  - No cherry picking is possible.
  - If we first generate the set S_1 with $|S_1| = n_1$ seeds and then generate the set S_2 with $|S_2| = n_2$ seeds with $n_2 > n_1$ for the same problem instance $\mathcal{I}$, then $S_1 \subset S_2$, i.e., we can iteratively increase the number of runs in an experiment.
- The seeds are still pseudo-random and uniformly distributed.

Advantages of this Method

- Random seeds are generated by a deterministic and replicable procedure:
  - No cherry picking is possible.
  - If we first generate the set S_1 with $|S_1| = n_1$ seeds and then generate the set S_2 with $|S_2| = n_2$ seeds with $n_2 > n_1$ for the same problem instance $\mathcal{I}$, then $S_1 \subset S_2$, i.e., we can iteratively increase the number of runs in an experiment.
- The seeds are still pseudo-random and uniformly distributed.
- Fairness.

Advantages of this Method

- Random seeds are generated by a deterministic and replicable procedure:
  - No cherry picking is possible.
  - If we first generate the set S_1 with $|S_1| = n_1$ seeds and then generate the set S_2 with $|S_2| = n_2$ seeds with $n_2 > n_1$ for the same problem instance $\mathcal{I}$, then $S_1 \subset S_2$, i.e., we can iteratively increase the number of runs in an experiment.
- The seeds are still pseudo-random and uniformly distributed.
- Fairness:
  - Apply two algorithms to the same instance $\mathcal{I}$?

Advantages of this Method

- Random seeds are generated by a deterministic and replicable procedure:
  - No cherry picking is possible.
  - If we first generate the set S_1 with $|S_1| = n_1$ seeds and then generate the set S_2 with $|S_2| = n_2$ seeds with $n_2 > n_1$ for the same problem instance $\mathcal{I}$, then $S_1 \subset S_2$, i.e., we can iteratively increase the number of runs in an experiment.
- The seeds are still pseudo-random and uniformly distributed.
- Fairness:
  - Apply two algorithms to the same instance $\mathcal{I}$?
  - They start with the same seeds.

Advantages of this Method

- Random seeds are generated by a deterministic and replicable procedure:
  - No cherry picking is possible.
  - If we first generate the set S_1 with $|S_1| = n_1$ seeds and then generate the set S_2 with $|S_2| = n_2$ seeds with $n_2 > n_1$ for the same problem instance $\mathcal{I}$, then $S_1 \subset S_2$, i.e., we can iteratively increase the number of runs in an experiment.
- The seeds are still pseudo-random and uniformly distributed.
- Fairness:
  - Apply two algorithms to the same instance $\mathcal{I}$?
  - They start with the same seeds.
  - If they use the same operators to generate their starting points, then they start with the same initial solutions.

Advantages of this Method

- Random seeds are generated by a deterministic and replicable procedure:
  - No cherry picking is possible.
  - If we first generate the set S_1 with $|S_1| = n_1$ seeds and then generate the set S_2 with $|S_2| = n_2$ seeds with $n_2 > n_1$ for the same problem instance $\mathcal{I}$, then $S_1 \subset S_2$, i.e., we can iteratively increase the number of runs in an experiment.
- The seeds are still pseudo-random and uniformly distributed.
- Fairness:
  - Apply two algorithms to the same instance $\mathcal{I}$?
  - They start with the same seeds.
  - If they use the same operators to generate their starting points, then they start with the same initial solutions.
  - Different algorithm performance will truly be due to the different decisions that the algorithms make, not a fluke due to different starting points...

Advantages of this Method

- Random seeds are generated by a deterministic and replicable procedure:
  - No cherry picking is possible.
  - If we first generate the set S_1 with $|S_1| = n_1$ seeds and then generate the set S_2 with $|S_2| = n_2$ seeds with $n_2 > n_1$ for the same problem instance $\mathcal{I}$, then $S_1 \subset S_2$, i.e., we can iteratively increase the number of runs in an experiment.
- The seeds are still pseudo-random and uniformly distributed.
- Fairness:
  - Apply two algorithms to the same instance $\mathcal{I}$?
  - They start with the same seeds.
  - If they use the same operators to generate their starting points, then they start with the same initial solutions.
  - Different algorithm performance will truly be due to the different decisions that the algorithms make, not a fluke due to different starting points...
- Different seeds for different instances avoids always using the same seeds, increases fairness.

Parallel and Distributed Runs

File and Directory Structure

- Assume that the base directory is "`results`".

File and Directory Structure

- Assume that the base directory is "`results`".
- One directory "`results/`$\mathrm{name}(\mathcal{A})$" is created for each algorithm setup $\mathcal{A}$, whereas $\mathcal{A}$ is user-defined, e.g., it could be "`rls_flip1`" for randomized local search (RLS) with a single-bit-flip operator.

File and Directory Structure

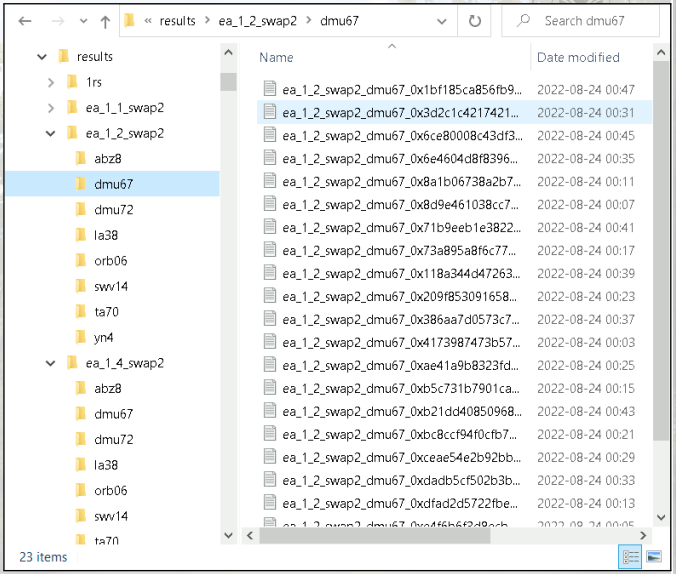
- Assume that the base directory is "`results`".
- One directory "`results/`$\mathrm{name}(\mathcal{A})$" is created for each algorithm setup $\mathcal{A}$, whereas $\mathcal{A}$ is user-defined, e.g., it could be "`rls_flip1`" for randomized local search (RLS) with a single-bit-flip operator.
- One directory "`results/`$\mathrm{name}(\mathcal{A})$`/`$\mathrm{name}(\mathcal{I})$" is created for each problem instance $\mathcal{I}$ to which algorithm setup $\mathcal{A}$ is applied.

File and Directory Structure

- Assume that the base directory is "`results`".
- One directory "`results/`$\mathrm{name}(\mathcal{A})$" is created for each algorithm setup $\mathcal{A}$, whereas $\mathcal{A}$ is user-defined, e.g., it could be "`rls_flip1`" for randomized local search (RLS) with a single-bit-flip operator.
- One directory "`results/`$\mathrm{name}(\mathcal{A})$`/`$\mathrm{name}(\mathcal{I})$" is created for each problem instance $\mathcal{I}$ to which algorithm setup $\mathcal{A}$ is applied.
- The file "`results/`$\mathrm{name}(\mathcal{A})$`/`$\mathrm{name}(\mathcal{I})$`/`$\mathrm{name}(\mathcal{A})$`_`$\mathrm{name}(\mathcal{I})$`_`$\mathrm{hex}(s)$`.tex`" is created for the run of algorithm setup $\mathcal{A}$ on problem instance $\mathcal{I}$ with random seed s.

File and Directory Structure

- The file "`results/`$\mathrm{name}(\mathcal{A})$`/`$\mathrm{name}(\mathcal{I})$`/`$\mathrm{name}(\mathcal{A})$`_`$\mathrm{name}(\mathcal{I})$`_`$\mathrm{hex}(s)$`.tex`" is created for the run of algorithm setup $\mathcal{A}$ on problem instance $\mathcal{I}$ with random seed s.



File and Directory Structure

- Assume that the base directory is "`results`".
- One directory "`results/`$\mathrm{name}(\mathcal{A})$" is created for each algorithm setup $\mathcal{A}$, whereas $\mathcal{A}$ is user-defined, e.g., it could be "`rls_flip1`" for randomized local search (RLS) with a single-bit-flip operator.
- One directory "`results/`$\mathrm{name}(\mathcal{A})$`/`$\mathrm{name}(\mathcal{I})$" is created for each problem instance $\mathcal{I}$ to which algorithm setup $\mathcal{A}$ is applied.
- The file "`results/`$\mathrm{name}(\mathcal{A})$`/`$\mathrm{name}(\mathcal{I})$`/`$\mathrm{name}(\mathcal{A})$`_`$\mathrm{name}(\mathcal{I})$`_`$\mathrm{hex}(s)$`.tex`" is created for the run of algorithm setup $\mathcal{A}$ on problem instance $\mathcal{I}$ with random seed s.

$\Rightarrow$ We have a clearly defined directory structure for experimental results.

File and Directory Structure

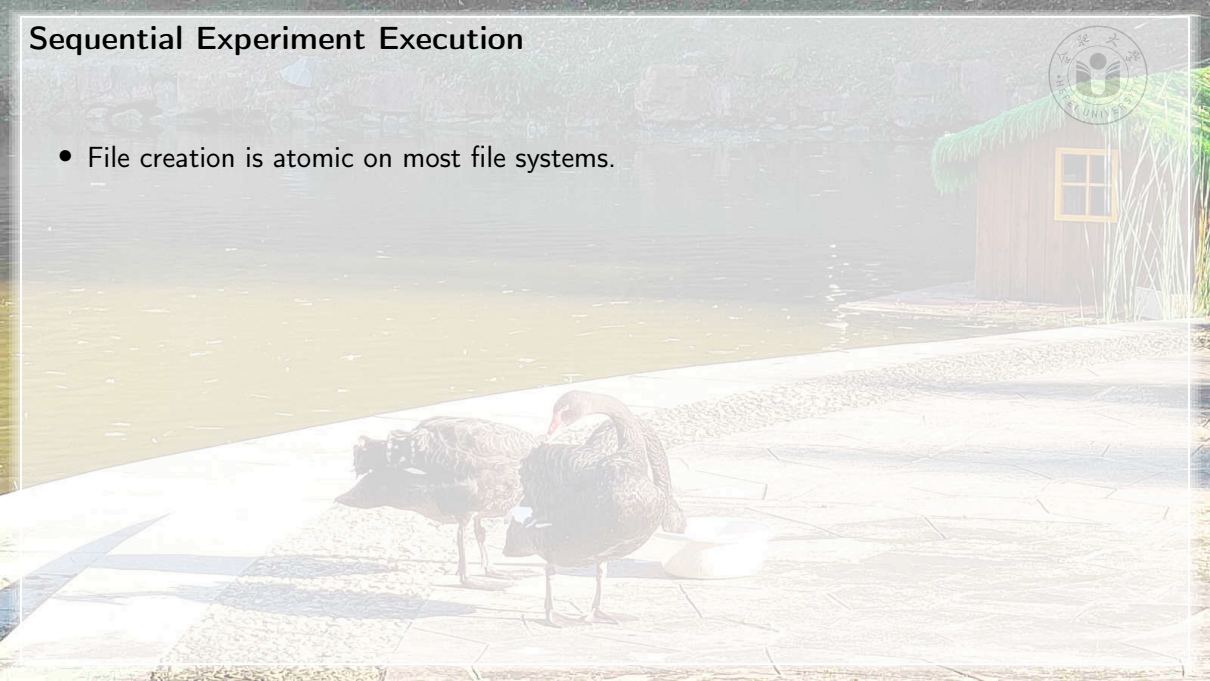
- Assume that the base directory is "`results`".
- One directory "`results/`$\mathrm{name}(\mathcal{A})$" is created for each algorithm setup $\mathcal{A}$, whereas $\mathcal{A}$ is user-defined, e.g., it could be "`rls_flip1`" for randomized local search (RLS) with a single-bit-flip operator.
- One directory "`results/`$\mathrm{name}(\mathcal{A})$`/`$\mathrm{name}(\mathcal{I})$" is created for each problem instance $\mathcal{I}$ to which algorithm setup $\mathcal{A}$ is applied.
- The file "`results/`$\mathrm{name}(\mathcal{A})$`/`$\mathrm{name}(\mathcal{I})$`/`$\mathrm{name}(\mathcal{A})$`_`$\mathrm{name}(\mathcal{I})$`_`$\mathrm{hex}(s)$`.tex`" is created for the run of algorithm setup $\mathcal{A}$ on problem instance $\mathcal{I}$ with random seed s.

$\Rightarrow$ We have a clearly defined directory structure for experimental results.

$\Rightarrow$ If the random seeds s are generated deterministically, then the same experiment will always yield the exactly same file names and directory structure *and* the file contents will also be the same (with the exception of stored clock time measurements, of course).

Sequential Experiment Execution

- File creation is atomic on most file systems.



Sequential Experiment Execution

- File creation is atomic on most file systems:
  - Creating a *new* file either succeeds or fails (e.g., if it already exits).

Sequential Experiment Execution

- File creation is atomic on most file systems:
  - Creating a *new* file either succeeds or fails (e.g., if it already exits).
  - If two processes or threads try to create a new file at the same time, it will succeed for only one process and fail for the other.

Sequential Experiment Execution

- File creation is atomic on most file systems:
  - Creating a *new* file either succeeds or fails (e.g., if it already exits).
  - If two processes or threads try to create a new file at the same time, it will succeed for only one process and fail for the other.
- In a sequential experiment, we execute the runs (algorithm setup $\mathcal{A}$ on problem instance $\mathcal{I}$ using random seed s) one after the other.

Sequential Experiment Execution

- File creation is atomic on most file systems:
  - Creating a *new* file either succeeds or fails (e.g., if it already exits).
  - If two processes or threads try to create a new file at the same time, it will succeed for only one process and fail for the other.
- In a sequential experiment, we execute the runs (algorithm setup $\mathcal{A}$ on problem instance $\mathcal{I}$ using random seed s) one after the other.
- Before doing a run, we try to create the (empty) corresponding log file.

Sequential Experiment Execution

- File creation is atomic on most file systems:
  - Creating a *new* file either succeeds or fails (e.g., if it already exits).
  - If two processes or threads try to create a new file at the same time, it will succeed for only one process and fail for the other.
- In a sequential experiment, we execute the runs (algorithm setup $\mathcal{A}$ on problem instance $\mathcal{I}$ using random seed s) one after the other.
- Before doing a run, we try to create the (empty) corresponding log file.
- The name and directory location of the files are deterministic and clear (see previous slide).

Sequential Experiment Execution

- File creation is atomic on most file systems:
  - Creating a *new* file either succeeds or fails (e.g., if it already exits).
  - If two processes or threads try to create a new file at the same time, it will succeed for only one process and fail for the other.
- In a sequential experiment, we execute the runs (algorithm setup $\mathcal{A}$ on problem instance $\mathcal{I}$ using random seed s) one after the other.
- Before doing a run, we try to create the (empty) corresponding log file.
- The name and directory location of the files are deterministic and clear (see previous slide).
- If this file creation fails (because the file already exist), we skip the run.

Sequential Experiment Execution

- File creation is atomic on most file systems:
  - Creating a *new* file either succeeds or fails (e.g., if it already exits).
  - If two processes or threads try to create a new file at the same time, it will succeed for only one process and fail for the other.
- In a sequential experiment, we execute the runs (algorithm setup $\mathcal{A}$ on problem instance $\mathcal{I}$ using random seed s) one after the other.
- Before doing a run, we try to create the (empty) corresponding log file.
- The name and directory location of the files are deterministic and clear (see previous slide).
- If this file creation fails (because the file already exist), we skip the run.
- Otherwise, we perform the run, collect all information in memory, and write it to the log file after the run completes.

Experiment Crashes

- What happens if an experiment crashes / power outage?

Experiment Crashes

- What happens if an experiment crashes / power outage?
- Delete all files of zero size.

Experiment Crashes

- What happens if an experiment crashes / power outage?
- Delete all files of zero size.
- Restart the experiment.

Experiment Crashes

- What happens if an experiment crashes / power outage?
- Delete all files of zero size.
- Restart the experiment.
- Only the incomplete runs are lost.

Experiment Crashes

- What happens if an experiment crashes / power outage?
- Delete all files of zero size.
- Restart the experiment.
- Only the incomplete runs are lost.
- No need to worry which parts completed and which need to be re-done.

Experiment Crashes

- What happens if an experiment crashes / power outage?
- Delete all files of zero size.
- Restart the experiment.
- Only the incomplete runs are lost.
- No need to worry which parts completed and which need to be re-done.
- The state is always clear: If a log file has non-zero size, the corresponding run has completed. Otherwise, it crashed.

Experiment Crashes

- What happens if an experiment crashes / power outage?
- Delete all files of zero size.
- Restart the experiment.
- Only the incomplete runs are lost.
- No need to worry which parts completed and which need to be re-done.
- The state is always clear: If a log file has non-zero size, the corresponding run has completed. Otherwise, it crashed.
- Since random seeds, log files, and their locations are deterministic, starting the same experiment twice will do the same runs and produce the same log files (skipping all completed runs...).

Seeds + Directory Structure = Parallelism!

- If we start the experiment program twice in parallel with the same root directory `results`, then both processes will try to create the same log files.

Seeds + Directory Structure = Parallelism!

- If we start the experiment program twice in parallel with the same root directory `results`, then both processes will try to create the same log files.
- The one that succeeds will do the run, the one that fails will try the next run, and so on.

Seeds + Directory Structure = Parallelism!

- If we start the experiment program twice in parallel with the same root directory `results`, then both processes will try to create the same log files.
- The one that succeeds will do the run, the one that fails will try the next run, and so on.
- Parallelism for free!

Seeds + Directory Structure = Parallelism!

- If we start the experiment program twice in parallel with the same root directory `results`, then both processes will try to create the same log files.
- The one that succeeds will do the run, the one that fails will try the next run, and so on.
- Parallelism for free!
- No inter-process communication, no exchange of data, no mutexes, no job queues, no synchronization ... the file system takes care of everything for us.

Seeds + Directory Structure = Distribution!

- We can share a directory over the (local) network.

Seeds + Directory Structure = Distribution!

- We can share a directory over the (local) network.
- Almost all operating systems support this out of the box.

Seeds + Directory Structure = Distribution!

- We can share a directory over the (local) network.
- Almost all operating systems support this out of the box.

Seeds + Directory Structure = Distribution!

- We can share a directory over the (local) network.
- Almost all operating systems support this out of the box.
- If file creation is atomic for a file system, then it is also atomic if the folder is shared, because the network share "sits on top" of the basic file system.

Seeds + Directory Structure = Distribution!

- We can share a directory over the (local) network.
- Almost all operating systems support this out of the box.
- If file creation is atomic for a file system, then it is also atomic if the folder is shared, because the network share "sits on top" of the basic file system.
- We create the root experiment folder `results`.

Seeds + Directory Structure = Distribution!

- We can share a directory over the (local) network.
- Almost all operating systems support this out of the box.
- If file creation is atomic for a file system, then it is also atomic if the folder is shared, because the network share "sits on top" of the basic file system.
- We create the root experiment folder `results`.
- We share the folder over two or more computers.

Seeds + Directory Structure = Distribution!

- We can share a directory over the (local) network.
- Almost all operating systems support this out of the box.
- If file creation is atomic for a file system, then it is also atomic if the folder is shared, because the network share "sits on top" of the basic file system.
- We create the root experiment folder `results`.
- We share the folder over two or more computers.
- We start the sane experiment program once for each CPU core in the shared folder on each machine.

Seeds + Directory Structure = Distribution!

- We can share a directory over the (local) network.
- Almost all operating systems support this out of the box.
- If file creation is atomic for a file system, then it is also atomic if the folder is shared, because the network share "sits on top" of the basic file system.
- We create the root experiment folder `results`.
- We share the folder over two or more computers.
- We start the sane experiment program once for each CPU core in the shared folder on each machine.
- Distributed computing for free!

Seeds + Directory Structure = Distribution!

- We can share a directory over the (local) network.
- Almost all operating systems support this out of the box.
- If file creation is atomic for a file system, then it is also atomic if the folder is shared, because the network share "sits on top" of the basic file system.
- We create the root experiment folder `results`.
- We share the folder over two or more computers.
- We start the sane experiment program once for each CPU core in the shared folder on each machine.
- Distributed computing for free!
- No cluster frameworks, no sockets, no communication needed ... all is done for us for free.

Summary

Summary

- moptipy allows us to conduct replicable self-documenting experiments with metaheuristic optimization algorithms in Python.

Summary

- moptipy allows us to conduct replicable self-documenting experiments with metaheuristic optimization algorithms in Python.
- These experiments can be executed in parallel on several processors or distributed over a network.

Summary

- moptipy allows us to conduct replicable self-documenting experiments with metaheuristic optimization algorithms in Python.
- These experiments can be executed in parallel on several processors or distributed over a network.
- moptipy ships with a wide range of experiment execution and statistical evaluation tools, including the ability to draw various charts.

Summary

- moptipy allows us to conduct replicable self-documenting experiments with metaheuristic optimization algorithms in Python.
- These experiments can be executed in parallel on several processors or distributed over a network.
- moptipy ships with a wide range of experiment execution and statistical evaluation tools, including the ability to draw various charts.
- It also offers implementation and benchmark datasets of many the classical $\mathcal{NP}$-hard tasks from Operations Research (OR).

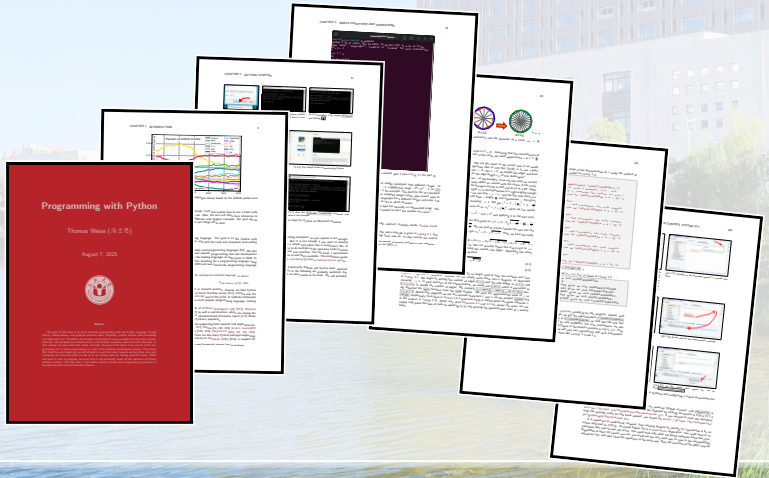
Summary

- moptipy allows us to conduct replicable self-documenting experiments with metaheuristic optimization algorithms in Python.
- These experiments can be executed in parallel on several processors or distributed over a network.
- moptipy ships with a wide range of experiment execution and statistical evaluation tools, including the ability to draw various charts.
- It also offers implementation and benchmark datasets of many the classical $\mathcal{NP}$-hard tasks from Operations Research (OR).
- If you are developing new algorithms, it is quite an ideal testbed to explore your methods!

Advertisement

Programming with Python

We have a freely available course book on *Programming with Python* at `https://thomasweise.github.io/programmingWithPython`, with focus on practical software development using the Python ecosystem of tools[66].

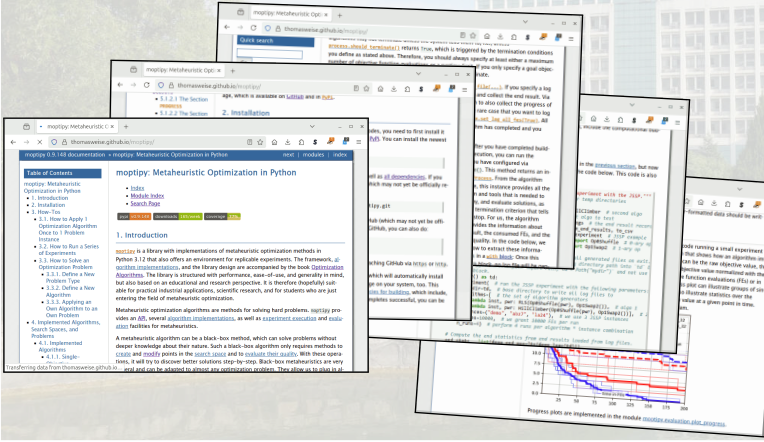


Databases

We have a freely available course book on *Databases* at `https://thomasweise.github.io/databases`, with actual practical examples using a real database management system (DBMS)[63].

Metaheuristic Optimization in Python: moptipy

We offer moptipy[69] a mature open source Python package for metaheuristic optimization, which implements several algorithms, can run self-documenting experiments in parallel and in a distributed fashion, and offers statistical evaluation tools.



谢谢您们！

Thank you!

Vielen Dank!

References I

[1] David Lee Applegate, Robert E. Bixby, Vašek Chvátal, and William John Cook. *The Traveling Salesman Problem: A Computational Study*. 2nd ed. Vol. 17 of Princeton Series in Applied Mathematics. Princeton, NJ, USA: Princeton University Press, 2007. ISBN: 978-0-691-12993-8 (cit. on p. 125).

[2] Thomas Bäck, David B. Fogel, and Zbigniew "Zbyszek" Michalewicz, eds. *Handbook of Evolutionary Computation*. Bristol, England, UK: IOP Publishing Ltd and Oxford, Oxfordshire, England, UK: Oxford University Press, 1997. ISBN: 978-0-7503-0392-7 (cit. on p. 122).

[3] Daniel J. Barrett. *Efficient Linux at the Command Line*. Sebastopol, CA, USA: O'Reilly Media, Inc., Feb. 2022. ISBN: 978-1-0981-1340-7 (cit. on p. 123).

[4] Jacek Błażewicz, Wolfgang Domschke, and Erwin Pesch. "The Job Shop Scheduling Problem: Conventional and New Solution Techniques". *European Journal of Operational Research* 93(1):1–33, Aug. 1996. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/0377-2217(95)00362-2 (cit. on p. 123).

[5] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (cit. on p. 123).

[6] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. Request for Comments (RFC) 8259. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Dec. 2017. URL: https://www.ietf.org/rfc/rfc8259.txt (visited on 2025-02-05) (cit. on p. 123).

[7] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler, eds. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. Wakefield, MA, USA: World Wide Web Consortium (W3C), Nov. 26, 2008–Feb. 7, 2013. URL: http://www.w3.org/TR/2008/REC-xml-20081126 (visited on 2024-12-15) (cit. on p. 126).

[8] Rainer E. Burkard, Eranda Çela, Panos Miltiades Pardalos, and Leonidas S. Pitsoulis. "The Quadratic Assignment Problem". In: *Handbook of Combinatorial Optimization*. Ed. by Panos Miltiades Pardalos, Ding-Zhu Du, and Ronald Lewis Graham. 1st ed. Boston, MA, USA: Springer, 1998, pp. 1713–1809. ISBN: 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_27 (cit. on p. 124).

[9] Eduardo Carvalho Pinto and Carola Doerr. *Towards a More Practice-Aware Runtime Analysis of Evolutionary Algorithms*. arXiv.org: Computing Research Repository (CoRR) abs/1812.00493. Ithaca, NY, USA: Cornell Universiy Library, Dec. 3, 2018. doi:10.48550/arXiv.1812.00493. URL: https://arxiv.org/abs/1812.00493 (visited on 2025-08-08). arXiv:1812.00493v1 [cs.NE] 3 Dec 2018 (cit. on p. 122).

References II

[10] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. "A Review of Machine Scheduling: Complexity, Algorithms and Approximability". In: *Handbook of Combinatorial Optimization*. Ed. by Panos Miltiades Pardalos, Ding-Zhu Du, and Ronald Lewis Graham. 1st ed. Boston, MA, USA: Springer, 1998, pp. 1493–1641. ISBN: 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_25. See also pages 21–169 in volume 3/3 by Norwell, MA, USA: Kluwer Academic Publishers. (Cit. on pp. 123, 126).

[11] Jiayang Chen (陈嘉阳), Zhize Wu (吴志泽), Sarah Louise Thomson, and Thomas Weise (汤卫思). "Frequency Fitness Assignment: Optimization Without Bias for Good Solution Outperforms Randomized Local Search on the Quadratic Assignment Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 27–37. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888600003837 (cit. on pp. 4–15, 123, 124).

[12] David Clinton and Christopher Negus. *Ubuntu Linux Bible*. 10th ed. Bible Series. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., Nov. 10, 2020. ISBN: 978-1-119-72233-5 (cit. on p. 125).

[13] Coding Gears and Train Your Brain. *YAML Fundamentals for DevOps, Cloud and IaC Engineers*. Birmingham, England, UK: Packt Publishing Ltd, Mar. 2022. ISBN: 978-1-80324-243-9 (cit. on p. 126).

[14] Timothy W. Cole and Myung-Ja K. Han. *XML for Catalogers and Metadata Librarians (Third Millennium Cataloging)*. 1st ed. Dublin, OH, USA: Libraries Unlimited, May 23, 2013. ISBN: 978-1-59884-519-8 (cit. on p. 126).

[15] Stephen Arthur Cook. "The Complexity of Theorem-Proving Procedures". In: *Third Annual ACM Symposium on Theory of Computing (STOC'1971)*. May 3–5, 1971, Shaker Heights, OH, USA. Ed. by Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman. New York, NY, USA: Association for Computing Machinery (ACM), 1971, pp. 151–158. ISBN: 978-1-4503-7464-4. doi:10.1145/800157.805047 (cit. on p. 126).

[16] "`csv` – CSV File Reading and Writing". In: *Python 3 Documentation*. *The Python Standard Library*. Beaverton, OR, USA: Python Software Foundation (PSF), 2001–2025. URL: https://docs.python.org/3/library/csv.html (visited on 2024-11-14) (cit. on p. 122).

[17] *Definition of Operations Research*. University of Western Ontario, London, ON, Canada: International Federation of Operational Research Societies (IFORS), 2020. URL: https://www.ifors.org/what-is-or (visited on 2026-01-01) (cit. on p. 124).

References III

[18] Justin Dennison, Cherokee Boose, and Peter van Rysdam. *Intro to NumPy*. Centennial, CO, USA: ACI Learning. Birmingham, England, UK: Packt Publishing Ltd, June 2024. ISBN: 978-1-83620-863-1 (cit. on p. 123).

[19] Ingy döt Net, Tina Müller, Pantelis Antoniou, Eemeli Aro, Thomas Smith, Oren Ben-Kiki, and Clark C. Evans. *YAML Ain't Markup Language (YAML™) version 1.2*. Revision 1.2.2. Seattle, WA, USA: YAML Language Development Team, Oct. 1, 2021. URL: https://yaml.org/spec/1.2.2 (visited on 2025-01-05) (cit. on p. 126).

[20] Stefan Droste, Thomas Jansen, and Ingo Wegener. "On the Analysis of the $(1 + 1)$ Evolutionary Algorithm". *Theoretical Computer Science* 276(1-2):51–81, Apr. 2002. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1016/S0304-3975(01)00182-7 (cit. on p. 122).

[21] Kelly Easton, George L. Nemhauser, and Michael A. Trick. "The Traveling Tournament Problem Description and Benchmarks". In: *7th International Conference on Principles and Practice of Constraint Programming (CP'01)*. Nov. 26–Dec. 1, 2001, Paphos, Cyprus. Ed. by Toby Walsh. Vol. 2239 of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2001, pp. 580–584. ISSN: 0302-9743. ISBN: 978-3-540-42863-3. doi:10.1007/3-540-45578-7_43 (cit. on p. 125).

[22] *ECMAScript Language Specification*. Standard ECMA-262, 3rd Edition. Geneva, Switzerland: Ecma International, Dec. 1999. URL: https://ecma-international.org/wp-content/uploads/ECMA-262_3rd_edition_december_1999.pdf (visited on 2024-12-15) (cit. on p. 123).

[23] Michael T. Goodrich. *A Gentle Introduction to NP-Completeness*. Irvine, CA, USA: University of California, Irvine, Apr. 2022. URL: https://ics.uci.edu/~goodrich/teach/cs165/notes/NPComplete.pdf (visited on 2025-08-01) (cit. on p. 126).

[24] Gregory Z. Gutin and Abraham P. Punnen, eds. *The Traveling Salesman Problem and its Variations*. Vol. 12 of Combinatorial Optimization (COOP). New York, NY, USA: Springer New York, May 2002. ISSN: 1388-3011. doi:10.1007/b101971 (cit. on p. 125).

[25] Charles R. Harris, K. Jarrod Millman, Stéfan van der Walt, Ralf Gommers, Pauli "pv" Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. "Array programming with NumPy". *Nature* 585:357–362, 2020. London, England, UK: Springer Nature Limited. ISSN: 0028-0836. doi:10.1038/S41586-020-2649-2 (cit. on p. 123).

References IV

[26] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (cit. on p. 123).

[27] Christian Heimes. "`defusedxml` 0.7.1: XML Bomb Protection for Python stdlib Modules". In: Mar. 8, 2021. URL: https://pypi.org/project/defusedxml (visited on 2024-12-15) (cit. on p. 126).

[28] Matthew Helmke. *Ubuntu Linux Unleashed 2021 Edition*. 14th ed. Reading, MA, USA: Addison-Wesley Professional, Aug. 2020. ISBN: 978-0-13-668539-5 (cit. on p. 125).

[29] John Hunt. *A Beginners Guide to Python 3 Programming*. 2nd ed. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (cit. on p. 124).

[30] *Information Technology – Universal Coded Character Set (UCS)*. International Standard ISO/IEC 10646:2020. Geneva, Switzerland: International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), Dec. 2020 (cit. on pp. 49–54, 121, 126).

[31] Robert Johansson. *Numerical Python: Scientific Computing and Data Science Applications with NumPy, SciPy and Matplotlib*. New York, NY, USA: Apress Media, LLC, Dec. 2018. ISBN: 978-1-4842-4246-9 (cit. on p. 123).

[32] Donald Ervin Knuth. *Fundamental Algorithms*. 3rd ed. Vol. 1 of The Art of Computer Programming. Reading, MA, USA: Addison-Wesley Professional, 1997. ISBN: 978-0-201-89683-1 (cit. on p. 125).

[33] Katie Kodes. *Intro to XML, JSON, & YAML*. London, England, UK: Payhip, 2019–Sept. 4, 2020 (cit. on p. 126).

[34] Tjalling C. Koopmans and Martin Beckmann. "Assignment Problems and the Location of Economic Activities". *Econometrica* 25(1):53–76, 1957. New Haven, CT, USA: The Econometric Society and Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0012-9682. doi:10.2307/1907742 (cit. on p. 124).

References V

[35] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. "Sequencing and Scheduling: Algorithms and Complexity". In: *Production Planning and Inventory*. Ed. by Stephen C. Graves, Alexander Hendrik George Rinnooy Kan, and Paul H. Zipkin. Vol. IV of Handbooks of Operations Research and Management Science. Amsterdam, The Netherlands: Elsevier B.V., 1993. Chap. 9, pp. 445–522. ISSN: 0927-0507. ISBN: 978-0-444-87472-6. doi:10.1016/S0927-0507(05)80189-6. URL: http://alexandria.tue.nl/repository/books/339776.pdf (visited on 2023-12-06) (cit. on pp. 123, 126).

[36] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester, West Sussex, England, UK: Wiley Interscience, Sept. 1985. ISSN: 0277-2698. ISBN: 978-0-471-90413-7 (cit. on p. 125).

[37] Kent D. Lee and Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (cit. on p. 124).

[38] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Jörg Lässig, Daan van den Berg, Sarah Louise Thomson, and Thomas Weise (汤卫思). "Addressing the Traveling Salesperson Problem with Frequency Fitness Assignment and Hybrid Algorithms". *Soft Computing* 28(17-18):9495–9508, July 2024. London, England, UK: Springer Nature Limited. ISSN: 1432-7643. doi:10.1007/S00500-024-09718-8 (cit. on pp. 4–15, 123, 125).

[39] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Jörg Lässig, Daan van den Berg, and Thomas Weise (汤卫思). "Solving the Traveling Salesperson Problem using Frequency Fitness Assignment". In: *IEEE Symposium Series on Computational Intelligence (SSCI'2022)*. Dec. 4–7, 2022, Singapore. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2022. ISBN: 978-1-6654-8769-6. doi:10.1109/SSCI51031.2022.10022296 (cit. on pp. 4–15, 123).

[40] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Matthias Thürer, Markus Wagner, and Thomas Weise (汤卫思). "Generating Small Instances with Interesting Features for the Traveling Salesperson Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 173–180. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888800003837 (cit. on pp. 4–15, 123).

References VI

[41] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter M. Hahn, and Tania Querido. "A Survey for the Quadratic Assignment Problem". *European Journal of Operational Research* 176(2):657–690, 2007. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2005.09.032 (cit. on p. 124).

[42] Mark Lutz. *Learning Python*. 6th ed. Sebastopol, CA, USA: O'Reilly Media, Inc., Mar. 2025. ISBN: 978-1-0981-7130-8 (cit. on p. 124).

[43] Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe, eds. *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0000195000003837.

[44] NumPy Team. *NumPy*. San Francisco, CA, USA: GitHub Inc and Austin, TX, USA: NumFOCUS, Inc. URL: https://numpy.org (visited on 2025-02-02) (cit. on p. 123).

[45] NumPy Team. "Permuted Congruential Generator (64-bit, PCG64)". In: *NumPy*. San Francisco, CA, USA: GitHub Inc and Austin, TX, USA: NumFOCUS, Inc., June 2022. URL: https://numpy.org/doc/2.4/numpy-ref.pdf (visited on 2025-12-28) (cit. on pp. 49–54, 124).

[46] Melissa E. O'Neill. *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation*. Tech. rep. HMC-CS-2014-0905. Claremont, CA, USA: Harvey Mudd College, Computer Science Department, Sept. 5, 2014. URL: https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf (visited on 2023-01-21) (cit. on pp. 49–54, 124).

[47] Panos Miltiades Pardalos, Ding-Zhu Du, and Ronald Lewis Graham, eds. *Handbook of Combinatorial Optimization*. 1st ed. Boston, MA, USA: Springer, 1998. ISBN: 978-1-4613-7987-4.

[48] Sanatan Rai and George Vairaktarakis. "NP-Complete Problems and Proof Methodology". In: *Encyclopedia of Optimization*. Ed. by Christodoulos A. Floudas and Panos Miltiades Pardalos. 2nd ed. Boston, MA, USA: Springer, Sept. 2008, pp. 2675–2682. ISBN: 978-0-387-74758-3. doi:10.1007/978-0-387-74759-0_462 (cit. on p. 126).

[49] Ernest E. Rothman, Rich Rosen, and Brian Jepson. *Mac OS X for Unix Geeks*. 4th ed. Sebastopol, CA, USA: O'Reilly Media, Inc., Sept. 2008. ISBN: 978-0-596-52062-5 (cit. on p. 123).

[50] Sartaj Sahni and Teofilo Gonzalez. "NP-complete Approximation Problems". *Journal of the ACM (JACM)* 23(3):555–565, 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0004-5411. doi:10.1145/321958.321975 (cit. on p. 124).

References VII

[51] *Secure Hash Standard (SHS)*. Federal Information Processing Standards Publication (FiPS PUB) 180-4. Gaithersburg, MD, USA: U.S. Department of Commerce, National Institute of Standards and Technology (NIST), Aug. 2015. doi:10.6028/NIST.FIPS.180-4. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (visited on 2025-12-28) (cit. on pp. 49–54, 125).

[52] Yakov Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. Request for Comments (RFC) 4180. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Oct. 2005. URL: https://www.ietf.org/rfc/rfc4180.txt (visited on 2025-02-05) (cit. on p. 122).

[53] Ellen Siever, Stephen Figgins, Robert Love, and Arnold Robbins. *Linux in a Nutshell*. 6th ed. Sebastopol, CA, USA: O'Reilly Media, Inc., Sept. 2009. ISBN: 978-0-596-15448-6 (cit. on p. 123).

[54] Bryan Sills, Brian Gardner, Kristin Marsicano, and Chris Stewart. *Android Programming: The Big Nerd Ranch Guide*. 5th ed. Reading, MA, USA: Addison-Wesley Professional, May 2022. ISBN: 978-0-13-764579-4 (cit. on p. 122).

[55] Drew Smith. *Modern Apple Platform Administration – macOS and iOS Essentials (2025)*. Birmingham, England, UK: Packt Publishing Ltd, Feb. 2025. ISBN: 978-1-80580-309-6 (cit. on p. 123).

[56] *The JSON Data Interchange Syntax*. Standard ECMA-404, 2nd Edition. Geneva, Switzerland: Ecma International, Dec. 2017. URL: https://ecma-international.org/publications-and-standards/standards/ecma-404 (visited on 2024-12-15) (cit. on p. 123).

[57] *The Unicode Standard, Version 15.1: Archived Code Charts*. South San Francisco, CA, USA: The Unicode Consortium, Aug. 25, 2023. URL: https://www.unicode.org/Public/15.1.0/charts/CodeCharts.pdf (visited on 2024-07-26) (cit. on p. 126).

[58] Sarah Louise Thomson, Gabriela Ochoa, Daan van den Berg, Tianyu Liang (梁天宇), and Thomas Weise (汤卫思). "Entropy, Search Trajectories, and Explainability for Frequency Fitness Assignment". In: *Parallel Problem Solving from Nature (PPSN XVIII)*. Vol. 1. Sept. 14–18, 2024, Hagenberg, Mühlkreis, Austria. Ed. by Michael Affenzeller, Stephan M. Winkler, Anna V. Kononova, Heike Trautmann, Tea Tušar, Penousal Machado, and Thomas Bäck. Vol. 15148 of Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer. ISSN: 0302-9743. ISBN: 978-3-031-70054-5. doi:10.1007/978-3-031-70055-2_23 (cit. on pp. 4–15, 123).

[59] Linus Torvalds. "The Linux Edge". *Communications of the ACM (CACM)* 42(4):38–39, Apr. 1999. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/299157.299165 (cit. on p. 123).

References VIII

[60] *Unicode®15.1.0*. South San Francisco, CA, USA: The Unicode Consortium, Sept. 12, 2023. ISBN: 978-1-936213-33-7. URL: https://www.unicode.org/versions/Unicode15.1.0 (visited on 2024-07-26) (cit. on p. 126).

[61] Sander van Vugt. *Linux Fundamentals*. 2nd ed. Hoboken, NJ, USA: Pearson IT Certification, June 2022. ISBN: 978-0-13-792931-3 (cit. on p. 123).

[62] Kristian Verduin, Sarah Louise Thomson, and Daan van den Berg. "Too Constrained for Genetic Algorithms. Too Hard for Evolutionary Computing. The Traveling Tournament Problem". In: *15th International Joint Conference on Computational Intelligence (IJCCI'23)*. Nov. 13–15, 2023, Rome, Italy. Ed. by Niki van Stein, Francesco Marcelloni, H. K. Lam, Marie Cottrell, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2023, pp. 246–257. ISSN: 2184-3236. ISBN: 978-989-758-674-3. doi:10.5220/0012192100003595 (cit. on p. 125).

[63] Thomas Weise (汤卫思). *Databases*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), 2025. URL: https://thomasweise.github.io/databases (visited on 2025-01-05) (cit. on pp. 109, 122).

[64] Thomas Weise (汤卫思). *Global Optimization Algorithms – Theory and Application*. self-published, 2009. URL: https://www.researchgate.net/publication/200622167 (visited on 2025-07-25) (cit. on p. 122).

[65] Thomas Weise (汤卫思). *jsspInstancesAndResults: Results, Data, and Instances of the Job Shop Scheduling Problem*. San Francisco, CA, USA: GitHub Inc, 2019–2020. URL: https://github.com/thomasWeise/jsspInstancesAndResults (visited on 2025-08-20) (cit. on pp. 4–15).

[66] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), 2024–2025. URL: https://thomasweise.github.io/programmingWithPython (visited on 2025-01-05) (cit. on pp. 4–15, 108, 124).

[67] Thomas Weise (汤卫思), Raymond Chiong, Jörg Lässig, Ke Tang (唐珂), Shigeyoshi Tsutsui, Wenxiang Chen (陈文祥), Zbigniew "Zbyszek" Michalewicz, and Xin Yao (姚新). "Benchmarking Optimization Algorithms: An Open Source Framework for the Traveling Salesman Problem". *IEEE Computational Intelligence Magazine (CIM)* 9(3):40–52, Aug. 2014. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1556-603X. doi:10.1109/MCI.2014.2326101 (cit. on p. 125).

References IX

[68] Thomas Weise (汤卫思), Xinlu Li (李新路), Yan Chen (陈岩), and Zhize Wu (吴志泽). "Solving Job Shop Scheduling Problems without using a Bias for Good Solutions". In: *Genetic and Evolutionary Computation Conference, Companion Volume*. July 10–14, 2021, Lille, France. Ed. by Krzysztof Krawiec. New York, NY, USA: Association for Computing Machinery (ACM), 2021, pp. 1459–1466. ISBN: 978-1-4503-8351-6. doi:10.1145/3449726.3463124 (cit. on pp. 4–15).

[69] Thomas Weise (汤卫思) and Zhize Wu (吴志泽). "Replicable Self-Documenting Experiments with Arbitrary Search Spaces and Algorithms". In: *Conference on Genetic and Evolutionary Computation (GECCO'2023), Companion Volume*. July 15–19, 2023, Lisbon, Portugal. Ed. by Sara Silva and Luís Paquete. New York, NY, USA: Association for Computing Machinery (ACM), 2023, pp. 1891–1899. ISBN: 979-8-4007-0120-7. doi:10.1145/3583133.3596306 (cit. on pp. 4–15, 110, 123).

[70] Thomas Weise (汤卫思), Zhize Wu (吴志泽), Xinlu Li (李新路), and Yan Chen (陈岩). "Frequency Fitness Assignment: Making Optimization Algorithms Invariant under Bijective Transformations of the Objective Function Value". *IEEE Transactions on Evolutionary Computation* 25(2):307–319, Apr. 2021. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2020.3032090 (cit. on pp. 4–15).

[71] Thomas Weise (汤卫思), Zhize Wu (吴志泽), Xinlu Li (李新路), Yan Chen (陈岩), and Jörg Lässig. "Frequency Fitness Assignment: Optimization without Bias for Good Solutions can be Efficient". *IEEE Transactions on Evolutionary Computation* 27(4):980–992, 2023. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2022.3191698 (cit. on pp. 4–15).

[72] Thomas Weise (汤卫思), Yuezhong Wu (吴越钟), Raymond Chiong, Ke Tang (唐珂), and Jörg Lässig. "Global versus Local Search: The Impact of Population Sizes on Evolutionary Algorithm Performance". *Journal of Global Optimization* 66(3):511–534, Feb. 2016. London, England, UK: Springer Nature Limited. ISSN: 0925-5001. doi:10.1007/s10898-016-0417-5 (cit. on p. 125).

[73] CAO Xiang (曹翔), Zhize Wu (吴志泽), Daan van den Berg, and Thomas Weise (汤卫思). "Randomized Local Search vs. NSGA-II vs. Frequency Fitness Assignment on The Traveling Tournament Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 38–49. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012891500003837 (cit. on pp. 4–15, 123, 125).

References X

[74] Kinza Yasar and Craig S. Mullins. *Definition: Database Management System (DBMS)*. Newton, MA, USA: TechTarget, Inc., June 2024. URL: https://www.techtarget.com/searchdatamanagement/definition/database-management-system (visited on 2025-01-11) (cit. on p. 122).

[75] François Yergeau. *UTF-8, A Transformation Format of ISO 10646*. Request for Comments (RFC) 3629. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Nov. 2003. URL: https://www.ietf.org/rfc/rfc3629.txt (visited on 2025-02-05). See Unicode and[30] (cit. on pp. 49–54, 126).

[76] Rui Zhao (赵睿), Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Daan van den Berg, Matthias Thürer, and Thomas Weise (汤卫思). "Randomized Local Search on the 2D Rectangular Bin Packing Problem with Item Rotation". In: *Genetic and Evolutionary Computation Conference (GECCO'2024)*. July 14–18, 2024, Melbourne, VIC, Australia. Ed. by Xiaodong Li and Julia Handl. New York, NY, USA: Association for Computing Machinery (ACM), 2024, pp. 235–238. ISBN: 979-8-4007-0494-9. doi:10.1145/3638530.3654139 (cit. on pp. 4–15, 123).

[77] Rui Zhao (赵睿), Zhize Wu (吴志泽), Daan van den Berg, Matthias Thürer, Tianyu Liang (梁天宇), Ming Tan (檀明), and Thomas Weise (汤卫思). "Randomized Local Search for Two-Dimensional Bin Packing and a Negative Result for Frequency Fitness Assignment". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 15–26. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888500003837 (cit. on pp. 4–15, 123).

Glossary I

$(1+1)$ EA The $(1+1)$ EA is a local search algorithm that retains the best solution x_c discovered so far during the search[9,20]. In each step, it applies a unary search operator to this best-so-far solution x_c and derives a new solution x_n. If the new solution x_n is *better or equally good* when compared with x_c, i.e., not worse, then it replaces it, i.e., is stored as the new x_c. If the search space are bit strings of length n, then the $(1+1)$ EA uses a unary search operator that flips each bit independently with probability m/n, where usually $m = 1$. This operator is the main difference to randomized local search (RLS). The $(1+1)$ EA is a special case of the $(\mu+\lambda)$ evolutionary algorithm ($(\mu+\lambda)$ EA) where $\mu = \lambda = 1$.

EA An *evolutionary algorithm* is a metaheuristic optimization method that maintains a population of candidate solutions, which undergo selection (where better solutions are chosen with higher probability) and reproduction (where mutation and recombination create a new candidate solution from one or two existing ones, repectively)[2,64].

$(\mu+\lambda)$ EA The $(\mu+\lambda)$ EA is an evolutionary algorithm (EA) where, in each generation, λ offspring solutions are generated from the current population of μ parent solutions. The offspring and parent populations are merged, yielding $\mu+\lambda$ solutions, from which then the best μ solutions are ratained to form the parent population of the next generation. If the search space is the bit strings of length n, then this algorithm usually applies a mutation operator flipping each bit independently with probability $1/n$.

Android is a common operating system for mobile phones[54].

CSV *Comma-Separated Values* is a very common and simple text format for exchanging tabular or matrix data[52]. Each row in the text file represents one row in the table or matrix. The elements in the row are separated by a fixed delimiter, usually a comma (","), sometimes a semicolon (";"). Python offers some out-of-the-box CSV support in the `csv` module[16].

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*[63].

DBMS A *database management system* is the software layer located between the user or application and the database (DB). The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB[74].

Glossary II

JavaScript JavaScript is the predominant programming language used in websites to develop interactive contents for display in browsers[22].

JSON *JavaScript Object Notation* is a data interchange format[6,56] based on JavaScript[22] syntax.

JSSP The *Job Shop Scheduling Problem*[4,35] is one of the most prominent and well-studied scheduling tasks. In a JSSP instance, there are k machines and m jobs. Each job must be processed once by each machine in a job-specific sequence and has a job-specific processing time on each machine. The goal is to find an assignment of jobs to machines that results in an overall shortest makespan, i.e., the schedule which can complete all the jobs in the shortest time. The JSSP is $\mathcal{NP}$-complete[10,35].

Linux is the leading open source operating system, i.e., a free alternative for Microsoft Windows[3,26,53,59,61]. We recommend using it for this course, for software development, and for research. Learn more at https://www.linux.org. Its variant Ubuntu is particularly easy to use and install.

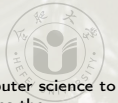
macOS or Mac OS is the operating system that powers Apple Mac(intosh) computers[49,55]. Learn more at https://www.apple.com/macos.

Microsoft Windows is a commercial proprietary operating system[5]. It is widely spread, but we recommend using a Linux variant such as Ubuntu for software development and for our course. Learn more at https://www.microsoft.com/windows.

moptipy is the *Metaheuristic Optimization in Python* library[69]. It has been used in several different research works, including[11,38–40,58,73,76,77]. Learn more at https://thomasweise.github.io/moptipy and https://thomasweise.github.io/moptipyapps.

NumPy is a fundamental package for scientific computing with Python, which offers efficient array datastructures[18,25,31]. Learn more at https://numpy.org[44].

Glossary III



OR Operations Research (or Operational Research) is the application of sciences such as mathematics and computer science to the management and organization of systems, organizations, enterprises, factories, or projects. It encompasses the development and application of problem-solving methods and techniques (such as mathematical optimization, simulation, queueing theory and other stochastic models) with the goal to improve decision-making and efficiency[17].

OS Operating System, the system that runs your computer, see, e.g., Linux, Microsoft Windows, macOS, and Android.

PCG A Permuted Congruential Generator (PCG) is a pseudo-random number generator[46]. Under NumPy, a 64-bit PCG is offered as a default method for generating seeding random number sequences[45]. You can import it via `from numpy.random import PCG64, Generator` and then create it from an `int` seed `h` via `Generator(PCG64(h))`.

Python The Python programming language[29,37,42,66], i.e., what you will learn about in our book[66]. Learn more at https://python.org.

QAP The *Quadratic Assignment Problem* is an optimization problem where the goal is to assign a set of n facilities to a set of n locations[8,11,34,41]. Such an assignment can be represented as a permutation x of the first n natural numbers, where x_i specifies the location where facility i should be placed. For each QAP, a distance matrix D is given, where D_{pq} specifies the distance from location p to location q, as well as a flow matrix F, where F_{ij} is the amount of material flowing from facility i to facility j. The objective function f then rates a permutation x as $f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} D_{x_i x_j} F_{ij}$. The QAP is $\mathcal{NP}$-complete[50].

RLS Randomized local search retains the best solution x_c discovered so far during the search and, in each step, it applies a unary search operator to this best-so-far solution x_c and derives a new solution x_n. If the new solution x_n is *better or equally good* when compared with x_c, i.e., not worse, then it replaces it, i.e., is stored as the new x_c. If the search space are bit strings of length n, then RLS uses a unary search operator that flips exactly one bit. This operator is the main difference to $(1+1)$ evolutionary algorithm ($(1+1)$ EA).

Glossary IV

SHA-512 SHA-512 is a cryptographic hash function belonging to the *Secure Hash Algorithm 2* family[51]. It is a one-way function that computes a checksum of 512 bits from data of arbitrary length. For the same data, the same checksum is computed. From the checksum, you cannot obtain the original data. Under Python, you can use `from hashlib import sha512` to get such a digest from a text string `s` via `bytearray(sha512(s.encode("utf8")).digest())`. Since the digest works with binary data, we first need to convert the text to such data. Here, we did this by encoding the string to UCS Transformation Format 8 (UCS).

stack trace A stack trace gives information the way in which one function invoked another. The term comes from the fact that the data needed to implement function calls is stored in a stack data structure[32]. The data for the most recently invoked function is on top, the data of the function that called is right below, the data of the function that called that one comes next, and so on. Printing a stack trace can be very helpful when trying to find out where an `Exception` occurred.

TSP In an instance of the *Traveling Salesperson Problem*, also known as *Traveling Salesman Problem*, a set of n cities or locations as well as the distances between them are defined[1,24,36,38,67,72]. The goal is to find the shortest round-trip tour that starts at one city, visits all the other cities one time each, and returns to the origin. The TSP is one of the most well-known $\mathcal{NP}$-hard combinatorial optimization problems[24].

TTP The *Traveling Tournament Problem* (TTP) is the combinatorial optimization problem of both efficiently and fairly organizing a tournament of n teams that play against each other in a pairwise fashion[21,73]. The efficient part boils down to arranging the games such that the total travel length is short, which is somewhat similar to the classical TSP. Initially, each team is at its home location. On each day, a team needs to travel if its scheduled game is not at its present location. On the last day, each team may need to travel back home unless their last game is a home game. The total travel length sums up the lengths of all travels over all teams. The fair part is represented in several constraints, such as doubleRoundRobin, compactness, maxStreak, and noRepeat. The TTP is $\mathcal{NP}$-hard[62].

Ubuntu is a variant of the open source operating system Linux[12,28]. We recommend that you use this operating system to follow this class, for software development, and for research. Learn more at https://ubuntu.com. If you are in China, you can download it from https://mirrors.ustc.edu.cn/ubuntu-releases.

UCS Universal Coded Character Set, see Unicode

Glossary V

Unicode A standard for assigning characters to numbers[30,57,60]. The Unicode standard supports basically all characters from all languages that are currently in use, as well as many special symbols. It is the predominantly used way to represent characters in computers and is regularly updated and improved.

UTF-8 The *UCS Transformation Format 8* is one standard for encoding Unicode characters into a binary format that can be stored in files[30,75]. It is the world wide web's most commonly used character encoding, where each character is represented by one to four bytes. It is backwards compatible with ASCII.

XML The *Extensible Markup Language* is a text-based language for storing and transporting of data[7,14,33]. It allows you to define elements in the form `<myElement myAttr="x">...text..</myElement>`. Different from comma-separated values (CSV), elements in XML can be hierarchically nested, like `<a><b><c>test</c></b><b>bla</b></a>`, and thus easily represent tree structures. XML is one of most-used data interchange formats. To process XML in Python, use the `defusedxml` library[27], as it protects against several security issues.

YAML *YAML Ain't Markup Language™* is a human-friendly data serialization language for all programming languages[13,19,33]. It is widely used for configuration files in the DevOps environment. See `https://yaml.org` for more information.

$\mathcal{NP}$ is the class of computational problems that can be solved in polynomial time by a non-deterministic machine and can be verified in polynomial time by a deterministic machine (such as a normal computer)[23].

$\mathcal{NP}$-complete A decision problem is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and all problems in $\mathcal{NP}$ are reducible to it in polynomial time[23,48]. A problem is $\mathcal{NP}$-complete if it is $\mathcal{NP}$-hard and if it is in $\mathcal{NP}$.

$\mathcal{NP}$-hard Algorithms that guarantee to find the correct solutions of $\mathcal{NP}$-hard problems[10,15,35] need a runtime that is exponential in the problem scale in the worst case. A problem is $\mathcal{NP}$-hard if all problems in $\mathcal{NP}$ are reducible to it in polynomial time[23].