



OPTAIN

Optimal Strategies to Retain Water and Nutrients

D5.1: Common optimisation protocol

Authors:

Michael Strauch (UFZ), Christoph Schürz (UFZ)

Delivery Date: 31. May 2024

This project has received funding from the European Union's
Horizon 2020 research and innovation programme under
Grant agreement No. 862756.



Disclaimer

This document reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

Intellectual Property Rights

© 2024, OPTAIN Consortium

All rights reserved.

This document contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation, or both.

This document is the property of the OPTAIN Consortium members. No copying or distributing in any form or by any means is allowed without the prior written agreement of the owner of the property rights. In addition to such written permission, the source must be clearly referenced.

Project Consortium



Document Information

Program	EU Horizon 2020 Research and Innovation Action H2020-EU.3.2.1.1 (SFS-23-2019)
Grant agreement No.	862756
Project acronym	OPTAIN
Project full name	Optimal strategies to retain and re-use water and nutrients in small agricultural catchments across different soil-climatic regions in Europe
Start of the project	September 2020
Duration	66 months
Project coordination	Prof. Dr. Martin Volk Helmholtz-Centre for Environmental Research GmbH - UFZ www.optain.eu
Deliverable	D5.1: Common optimisation protocol The main objective of OPTAIN's task 5.1 was to enable catchment-scale modellers to run a multi-objective optimisation of the allocation and combination of Natural/Small Water Retention Measures (NSWRM) in their own case study. This report (i) introduces the Pareto optimal NSWRM implementation plans as one of the project's key products, (ii) describes OPTAIN's optimisation concept, (iii) outlines the requirements that a SWAT+ model setup must meet before it can be used for the optimisation, (iv) shows how to build a SWATmeasR project as a key tool for implementing NSWRM in a SWAT+ model, and (v) provides a protocol on how to run the optimisation using the software CoMOLA.
Work package	WP5: Optimisation of NSWRM plans
Task	Task 5.1: Empower cases studies with spatial optimisation skills
Lead beneficiary	Helmholtz Centre for Environmental Research GmbH - UFZ
Author(s)	Michael Strauch (UFZ), Christoph Schürz (UFZ)
Contributor(s)	
Quality check	Felix Witing (UFZ), Martin Volk (UFZ)
Planned delivery date	Month 45 (May 2024)
Actual delivery date	31/05/2024
Citation	Strauch, M., Schürz, C. (2024): <i>Common optimisation protocol</i> . Deliverable D4.5 EU Horizon 2020 OPTAIN Project, Grant agreement No. 862756
Dissemination level*	PU

*PU = Public; PP = Restricted to other program participants (including the Commission Services); CO = Confidential, only for members of the consortium (including the Commission Services).

Deliverable status

Version	Date	Author(s)/Contributor(s)	Notes
0.5	07.05.2024	Michael Strauch (UFZ), Christoph Schürz (UFZ), Felix Witing (UFZ)	Optimisation workshop for OPTAIN CS SWAT+ modellers in Leipzig & first draft of optimisation protocol.
0.8	27.05.2024	Michael Strauch (UFZ), Christoph Schürz (UFZ)	First complete draft
0.9	30.05.2024	Felix Witing (UFZ), Martin Volk (UFZ)	Revision
1.0	31.05.2024	Michael Strauch (UFZ), Christoph Schürz (UFZ)	Final

Summary

The objective of this deliverable D5.1 is to enable catchment-scale modellers to perform a multi-objective optimisation of the allocation and combination of Natural/Small Water Retention Measures (NSWRMs) in their own case study (CS).

This report (i) introduces the Pareto optimal NSWRM implementation plans as one of the project's key products, (ii) describes OPTAIN's optimisation concept, (iii) outlines the requirements that a SWAT+ model setup must meet before it can be used for the optimisation, (iv) shows how to build a SWATmeasR project as a key tool for implementing NSWRMs in a SWAT+ model, and (v) provides a protocol on how to run the optimisation using the software CoMOLA.

The report should also be useful beyond the OPTAIN project for interested SWAT+ modellers who wish to use their model to optimise spatially explicit NSWRM or Best Management Practice (BMP) plans against multiple catchment-scale objectives.

Table of Contents

Summary	5
Abbreviation list.....	7
1. Introduction.....	8
1.1. Objective.....	8
2. OPTAIN's optimisation concept.....	9
2.1. Definition of optimisation objectives	10
2.2. Definition of the decision space	11
2.3. Functions evaluating the objectives.....	11
2.4. Multi-objective optimisation algorithm.....	12
3. SWAT+ model preparation.....	14
3.1. Basic SWAT+ model configuration	14
3.2. Building a SWATmeasR project.....	15
3.2.1. General SWATmeasR workflow	15
3.2.2. Initialising a new SWATmeasR project	17
3.2.3. General structure of a SWATmeasR project	18
3.2.4. NSWRM definition.....	24
3.2.5. Definition of NSWRM locations.....	32
3.2.6. NSWRM implementation	33
4. Running the optimisation	36
4.1. python installation.....	36
4.1.1. miniconda installation.....	36
4.1.2. miniconda setup	38
4.2. CoMOLA file structure.....	41
4.3. Setting up and starting an optimisation run.....	42
4.3.1. Add your own txt folder	43
4.3.2. Adjust the <i>SWAT.R</i> script	43
4.3.3. Configure the master file (<i>config.ini</i>).....	47
4.3.4. Starting the optimisation (<i>run_comola.bat</i>)	48
4.4. Analysis of results.....	49
5. References.....	52

Abbreviation list

BMP	Best Management Practice
COCOA	Contiguous Object COnnectivity Approach
CoMOLA	Constrained Multi-objective Optimisation of Land use Allocation
CS	Case Study
EPI	Environmental Performance Indicator
HRUs	Hydrologic Response Units
MARG	Multi-Actor Reference Group
NSWRMs	Natural/Small Water Retention Measures
OPTAIN	OPTimal strategies to retAIN and re-use water and nutrients in small agricultural catchments across different soil-climatic regions in Europe
SPI	Socio-economic Performance Indicator
SWAT	Soil and Water Assessment Tool
SWAT+	Soil and Water Assessment Tool Plus
TN	Total Nitrogen
TP	Total Phosphorus
WP	Work Package

1. Introduction

1.1. Objective

The objective of Task 5.1 of the OPTAIN project is to explore where to implement which measure(s) within each of the case studies in order to best possible meet various environmental and socio-economic objectives. Based on previous Multi-Actor Reference Group (MARG) workshops, we have learned which water and nutrient retention problems are most relevant in the different case studies and have identified a number of promising Natural/Small Water Retention Measures (NSWRMs) that we can model at individual sites using the SWAT+ model in combination with the Contiguous Object COnnectivity Approach (COCO) developed in this project (Schürz et al., 2022). We have run a set of model scenarios, where we evaluated the effectiveness of individual NSWRMs, considering all sites where the implementation appeared reasonable (Piniewski et al., 2024). The evaluation was based on a wide range of environmental performance indicators (EPIs) relating to water and nutrient retention and crop yield. We also developed and applied functions to calculate a number of socio-economic performance indicators - SPIs (such as agricultural gross margin and implementation costs), which are described in detail in deliverable D2.2 (Krzeminska & Monaco, 2022) and in the forthcoming deliverable D4.5 ('Attractiveness and socio-economic assessment of NSWRMs'). What is still missing, and will be addressed in WP 5, is an integrative assessment (a cost-benefit analysis) that takes into account both the most relevant EPIs and the most relevant SPIs, while at the same time allowing for any individual allocation and combination of the NSWRMs at hand. In a particular location within the catchment, some NSWRMs may be more efficient at retaining water and nutrients than others. At other sites, a combination of different NSWRMs may be most effective. Any NSWRM implementation is likely to involve trade-offs (e.g. with agricultural production and implementation costs), and local site-level effects may not translate linearly to the catchment level. Given the large number of possible implementation sites for the set of different NSWRMs (~3-6 depending on the case study), there could be millions of different combinations (hereafter referred to as 'NSWRM plans'), each with different environmental and socio-economic performance.

In order to explore the best possible NSWRM plans for different, sometimes conflicting objectives, the SWAT+ models built in each case study need to be coupled with an efficient search algorithm. In OPTAIN, we use the Non-dominated Sorting Genetic Algorithm II (NSGA-II), a widely used Pareto-based optimisation algorithm (Deb et al., 2002), which is embedded in the Constrained Multi-objective Optimisation of Land use Allocation (CoMOLA) tool, developed at UFZ (Strauch et al., 2019). The result of such a search is not a single optimal NSWRM plan, but a large number of Pareto optimal solutions (Figure 1.1). Pareto optimal solutions are solutions for which no objective can be further improved without compromising at least one of the other objectives. From such a set of best alternatives (with minimal trade-offs between the objectives), decision makers can discuss and select appropriate solutions according to their preferences (Cord et al., 2017, Kaim et al.,

2020). In a later task of work package (WP) 5, each case study will conduct such a preference analysis with its MARG (Task 5.4). The ultimate goal of this Task 5.1 is to enable CS modellers to conduct their own multi-objective optimisation of NSWRM plans. As a result, each case study should be able to run its SWAT+ model within CoMOLA to identify NSWRM plans that are close to Pareto optimality. The following chapters were discussed at a 3-day workshop with the OPTAIN CS modellers and (partially) applied in practice.

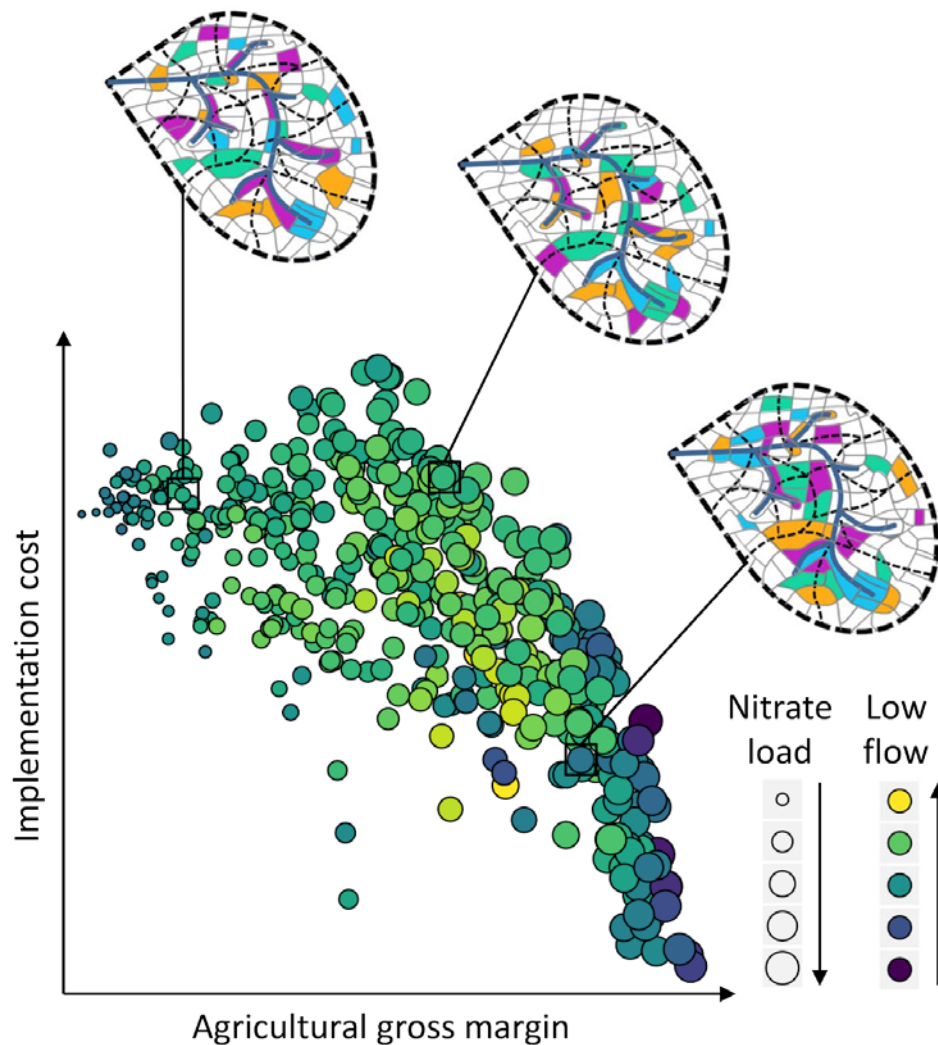


Figure 1.1: Schematic illustration of Pareto optimal NSWRM implementation plans

2. OPTAIN's optimisation concept

Multi-objective optimisation of management options requires four components, regardless of its purpose:

- (1) The definition of optimization objectives,

- (2) the definition of the decision space,
- (3) functions evaluating the objectives based on the decisions, and
- (4) a multi-objective optimisation algorithm.

In the following, we briefly describe how these components are defined in OPTAIN to optimise NSWRM plans.

2.1. Definition of optimisation objectives

First and foremost, we have to define the objectives (or goals) for which the NSWRM plans are to be optimised. OPTAIN's WP2 defined a number of different environmental and socio-economic performance indicators (EPIs/SPIs) that could be used as optimisation objectives. The multi-objective optimisation algorithm implemented in CoMOLA (NSGA-II) can handle up to four objectives. Ideally, NSWRM plans should solve or minimise the most relevant water and nutrient related problem(s) of the case study at minimum cost to farmers and society. Therefore, the set of optimisation objectives should consist of the following indicators:

- (1) One EPI that is addressing the most relevant water related problem (e.g. soil water content or a river discharge indicator for a specific period within the year),
- (2) Another EPI that is addressing the most relevant nutrient related problem (e.g. nitrogen, phosphorus or sediment concentrations or loads at the catchment outlet). If there is no nutrient related problem in the case study or if SWAT+ has not been calibrated for nutrients, a second water related EPI could be selected.
- (3) One SPI that is indicating the catchment's agricultural production (e.g. the sum of all crop-specific yields expressed as grain units or the agricultural gross margin).
- (4) Another SPI that is referring to implementation and maintenance costs of NSWORMs, with or without considering subsidies. Agricultural production, NSWORM costs and subsidies can also be combined in one indicator if appropriate. This would leave room for one more indicator to be considered as optimisation objective

It is also possible to integrate several indicators into one objective function. For example, if total nitrogen (TN) and total phosphorus (TP) loads were to be used together as one water quality objective, TN and TP loads could simply be summed. However, as TN loads are naturally higher than TP loads, TN loads would be over-emphasised. Appropriate weighting may therefore be required. The definition of performance indicators as optimisation objectives is a crucial step for each case study. Section 4.3 shows in more detail how to define the objectives within the CoMOLA workflow.

The calculation of the socio-economic optimisation objectives is based on selected input files (e.g. parameterization of the agricultural management) and output files (e.g. crop yield simulations) of the SWAT+ model, as well as a set of CS-specific economic parameters. In practice, the calculation of the SPIs is also done using an R script in the postprocessing of a SWAT+ run. Details on the calculation of the socio-economic optimisation objectives are described in OPTAIN's in deliverables D2.2 and D4.5.

2.4. Multi-objective optimisation algorithm

Multi-objective optimisation algorithms are increasingly used for land use allocation problems in agricultural landscapes, as they are able to approximate Pareto optimal solutions from a large number of possible land use/land management configurations (Kaim et al., 2018; Memmah et al., 2015). OPTAIN uses NSGA-II (Deb et al., 2002), a popular multi-objective genetic algorithm included in the CoMOLA software. The step-by-step process of this algorithm is as follows (see also Figure 2.2):

Step 1: Initialization

Generate Initial Population: Create an initial population P_0 of N individuals (i.e. individual NSWRM plans) randomly within the feasible¹ solution space. Population size N is an important parameter to be adjusted by case studies (section 4.3).

Evaluate Population: Calculate the fitness values of each individual based on the objective functions. In other words, run SWAT+ for each NSWRM plan of the initial population and calculate the value of the optimisation objectives based on the SWAT+ outputs and the socio-economic parameters (section 2.1).

Step 2: Non-dominated Sorting

Rank Individuals: Sort the population into different fronts using non-dominated² sorting. Front F_1 consists of non-dominated individuals, F_2 consists of individuals dominated only by those in F_1 , and so on.

Crowding Distance Assignment: Calculate the crowding distance³ for each individual in the population. This metric helps maintain diversity by favouring individuals in less crowded regions of the solution space.

¹ In OPTAIN, maximum implementation scenarios of single NSWRM have been co-created with local stakeholders. We therefore assume that all possible NSWRM plans are feasible, although some NSWRM combinations may in fact be unrealistic. Feasibility of NSWRM combinations will be addressed at a later stage when preferred solutions are identified with local stakeholders from the final set of Pareto optimal NSWRM plans. By assuming feasibility of all NSWRM plans during the optimisation, computationally intensive constraint handling is avoided.

² An individual A dominates another individual B if A is no worse than B in all objectives and better in at least one objective.

³ For each individual, the crowding distance is calculated based on the average distance of the two neighbouring individuals on either side along each objective. This distance measures how close an individual is to its neighbours.

Step 3: Selection

Tournament Selection: Use binary tournament selection based on rank and crowding distance to select parent individuals for the next generation. An individual with a lower rank is preferred, and if ranks are equal, the one with a higher crowding distance is preferred.

Step 4: Create and Evaluate Offspring Population

The offspring population Q_t of size N is generated using crossover and mutation from the selected parents.

Crossover: Apply crossover (recombination) operators to selected parents to generate offspring. Common crossover methods include single-point, multi-point, and simulated binary crossover (SBX).

Mutation: Apply mutation operators to offspring to introduce variability. This can involve bit-flip mutations for binary representations or polynomial mutations for real-valued representations.

Evaluate Offspring Population: Calculate the fitness values of each individual based on the objective functions. In other words, run SWAT+ for each NSWRM plan of the offspring population and calculate the value of the optimisation objectives based on the SWAT+ outputs and the socio-economic parameters (section 2.1).

Step 5: Combine Populations

Combine the parent population P_t and offspring population Q_t to form a combined population R_t of size $2N$.

Step 6: Non-dominated Sorting of Combined Population

Perform non-dominated sorting on the combined population R_t to identify the different fronts F_1, F_2, \dots

Step 7: Selection of New Population

Start with an empty new population P_{t+1} . Add individuals from each front F_i to P_{t+1} until the population size exceeds N .

If adding all individuals from front F_i causes the population size to exceed N , sort the individuals in F_i by their crowding distance in descending order and select the top individuals to fill the population up to size N .

Step 8: Loop Until Termination

Repeat the process from step 3 to step 7 for a predefined number of generations or until a convergence criterion is met. In OPTAIN, case studies need to define the number of generations (section 4.3). The core principles of this loop are shown in Figure 2.2.

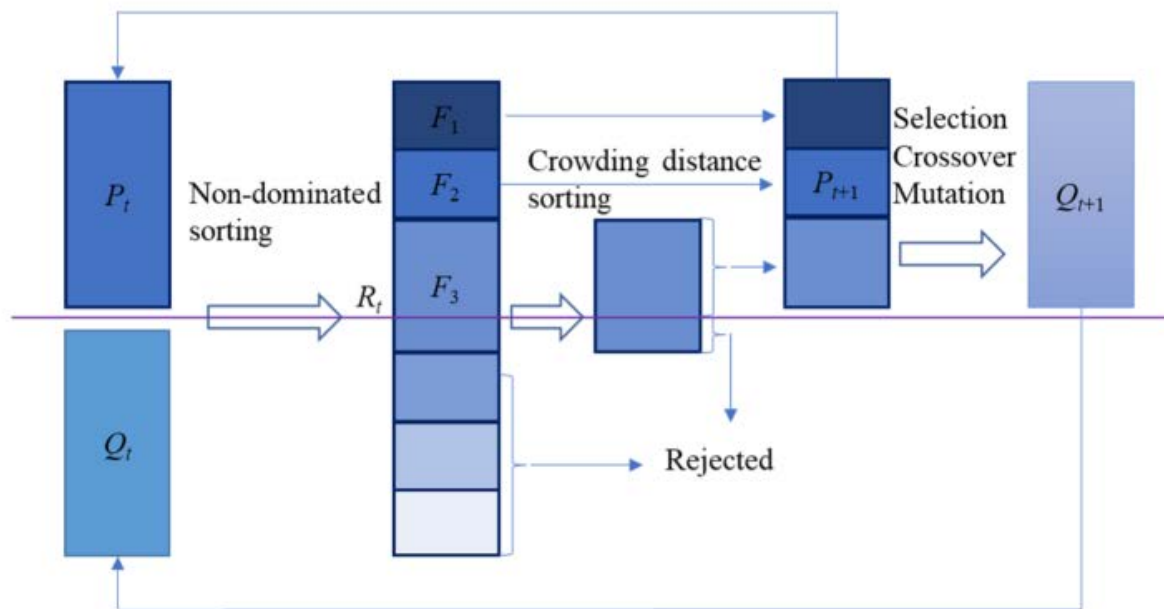


Figure 2.2 (source: Jiang et al., 2021): Core principles of the NSGA-II: Apply non-dominated sorting to the combined (parent and offspring) population. If necessary, use crowding distance to cut off the sorted combined population to population size N (forming the new parent population). Apply tournament selection, crossover and mutation to the new parents to form a new offspring population and calculate its fitness values. The procedure is repeated until a termination criterion is met.

3. SWAT+ model preparation

3.1. Basic SWAT+ model configuration

All SWAT+ modelling tasks in OPTAIN strongly build on the preceding steps in the modelling workflow. This is particularly true for the NSWRM implementation in the SWAT+ model setups and eventually for the optimization of NSWRM combinations. Based on the decisions made for a SWAT+ model setup in OPTAIN the R package SWATmeasR has been developed to enable an automatable approach for the implementation of NSWRMs in SWAT+ model setups. The conceptual approach on how specific NSWRMs are represented in SWAT+ model setups has been defined by Marval et al. (2022). To enable the implementation of structural NSWRMs in particular, the SWAT+ model setups are required to represent a great detail of connection of water and nutrient fluxes in the landscape. In OPTAIN we therefore developed the contiguous object connectivity approach (COCOA) and implemented this concept in the newly developed SWAT+ model builder SWATbuildR (Schürz, 2024a) which must be used by every OPTAIN case study to set up their SWAT+ models. Farm management operation schedules were implemented in every case study model setup with the use of the R package SWATfarmR (Schürz, 2024b). In order to analyse the effects of management-related NSWRMs, different management plans were set up for each field unit in a model

that represents the status quo and a possible scenario. For an analysis, the individual management schedules must be interchangeable to switch between them.

Only SWAT+ model setups that follow the OPTAIN model setup guidelines can use the full functionality for the implementation of NSWORMs with SWATmeasR which is documented below. A detailed guideline on the required model setup procedure is documented in Schürz et al. (2022). Nevertheless, SWAT+ model setups created with different approaches (e.g. using QSWAT+) can use a limited set of NSWORM types to be implemented with SWATmeasR. The implementation of land use change type NSWORMs (such as any greening measure or afforestation) does not necessarily require any specific configuration of the model's land object connectivity. Also, the addition of wetland water storages to HRUs can be done with SWATmeasR independent of the land object connectivity (if water routing of a land object is not changed). The implementation of structural measures, such as ponds, utilises the COCOA approach when implemented in a model setup and therefore requires the SWAT+ model to be built with SWATbuildR. The implementation of farm management related NSWORMs rely on SWATfarmR projects in the current form of SWATmeasR. However, this can be generalised to the use of management.sch input files in a later version of SWATmeasR.

3.2. Building a SWATmeasR project

SWATmeasR is an R package (<https://git.ufz.de/schuerz/swatmeasr>) for a fast and easy implementation of NSWORMs in SWAT+ model setups. SWATmeasR enables the implementation of selected NSWORMs, such as land use change, changes in the farm management, or structural measures such as ponds and wetlands. With SWATmeasR, NSWORMs are defined by specific parameters and their "locations" in a model setup. All defined NSWORMs can be easily combined and implemented in the corresponding SWAT+ model setup.

3.2.1. General SWATmeasR workflow

The SWATmeasR workflow can be separated into two major parts, the setup of a SWATmeasR project and the implementation of NSWORMs with an existing SWATmeasR project. Figure 3.1 provides a general overview of the workflows in the setup and the implementation phase.

The setup of a SWATmeasR project involves three steps: (i) the initialization of a new SWATmeasR project with the function `new_measr()`, (ii) loading the definitions (parameterizations) of all NSWORMs that should be implemented in the model setup with `measr_project$load_nswrm_definition()` and (iii) the definition of the locations of all NSWORMs that should be implemented with `measr_project$load_nswrm_location()`.

The NSWORM implementation is exemplary illustrated in Figure 3.1 for the implementation of measures in the CoMOLA workflow. However, the NSWORM implementation procedure is the same in any modelling workflow in which the effects of NSWORMs are to be simulated. NSWORMs are implemented in the SWAT+

model with the function `measr_project$implement_nswrm()`, which implements the selected NSWORMs in the SWAT+ input tables in R. To write the changes in the SWAT+ text input files into the SWAT+ project folder, the function `measr_project$write_swat_inputs()` must be called. Once the NSWORMs have been implemented, model simulations are carried out and the results are analysed to examine the effects of the implemented NSWORMs. The initial state of the SWAT+ text input files in the project folder and in the corresponding tables in R is restored by calling the function `measr_project$reset()`.

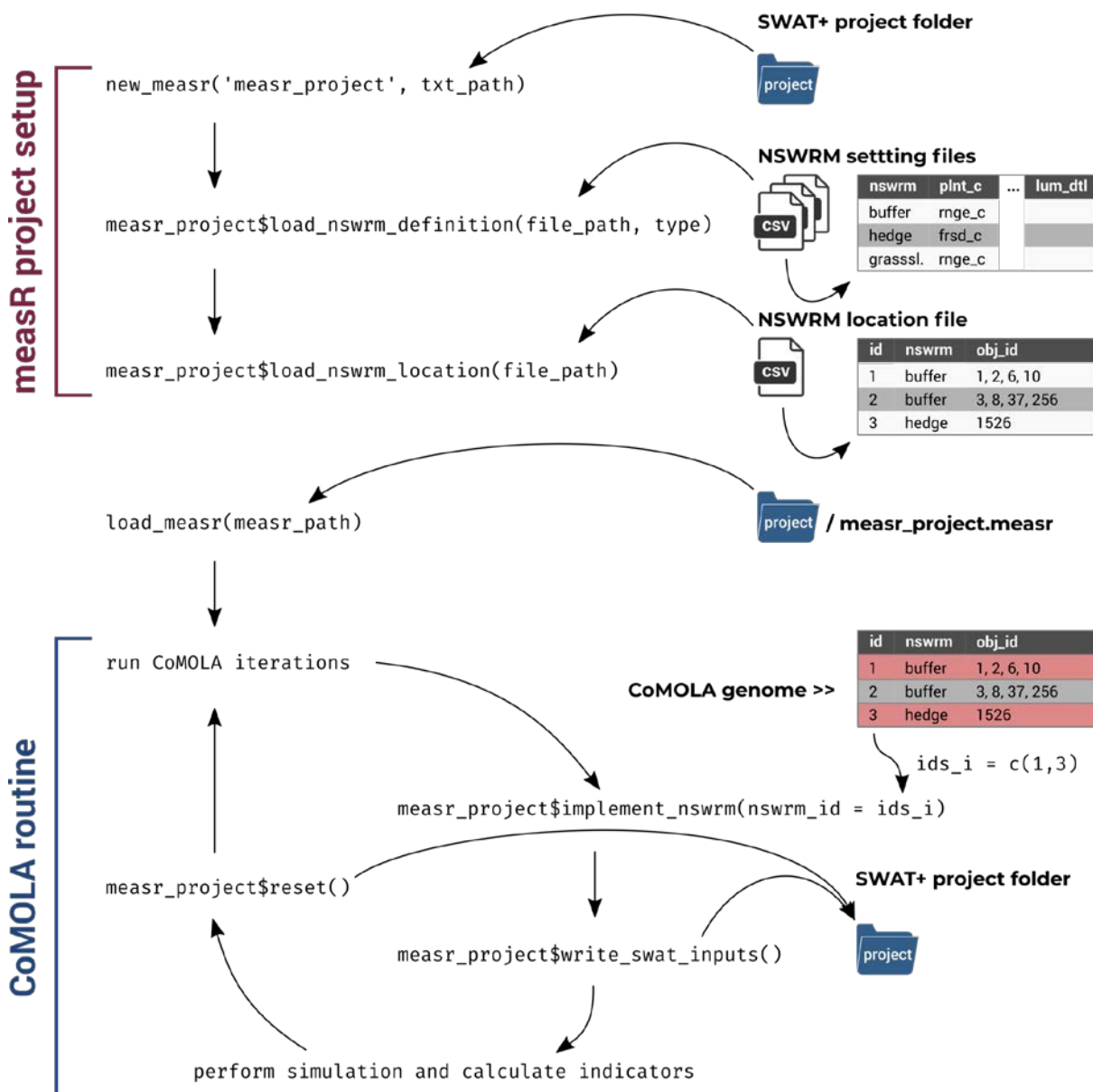


Figure 3.1: Overview of the SWATmeasR workflow.

3.2.2. Initialising a new SWATmeasR project

SWATmeasR is based on the R6 object class (Chang, 2022), which is an implementation of object-oriented programming in R. In simple words, R6 objects can contain data and functions which, when executed, can modify the data stored in the object. The following code generates a new SWATmeasR project that creates such an object in R, which stores data and provides functions to apply to the stored data.

```
new_measr(project_name = 'measr_project',  
          project_path = 'Path:/to/your/SWAT/txt/folder')
```

The function `new_measr()` initialises a new SWATmeasR project (in the following simply called measr project) with the name that was provided by the input argument `project_name`. The input argument `project_path` defines the path of the SWAT+ project folder on the hard drive for which the new SWATmeasR project should be generated. The initialization of the new SWATmeasR project reads all relevant input text files from the SWAT+ project folder in `project_path` and stores them in an object with the `project_name` in the R environment. Additionally to the SWAT+ input tables, the new SWATmeasR provides a set of functions that will be used later to e.g. load additional data or implement NSWRRMs. At the same time when the R object is generated, a *.measr file with the name “`project_name.measr`” is saved in the SWAT+ project folder. It contains the same data as the object in the R environment and is essentially a backup of the project in R, which can be loaded into R with the function `load_measr()`. Each time the SWATmeasR project is modified in R, also the *.measr file on the hard drive is written and updated.

In the example in Figure 3.2, a SWATmeasR project was initialised with the `project_name = "schoeps_240502"`. When calling the function `new_measr()` it prints that the SWAT+ input files were read from the SWAT+ project folder defined by `project_path` (Figure 3.2a). Figure 3.2 a) also shows the list of SWAT+ input tables, which were read and saved in the new R object `schoeps_240502`. The new SWATmeasR project is present in the R environment (Figure 3.2b) and a file with the name “`schoeps_240502.measr`” was written into the SWAT+ project folder (Figure 3.2c).

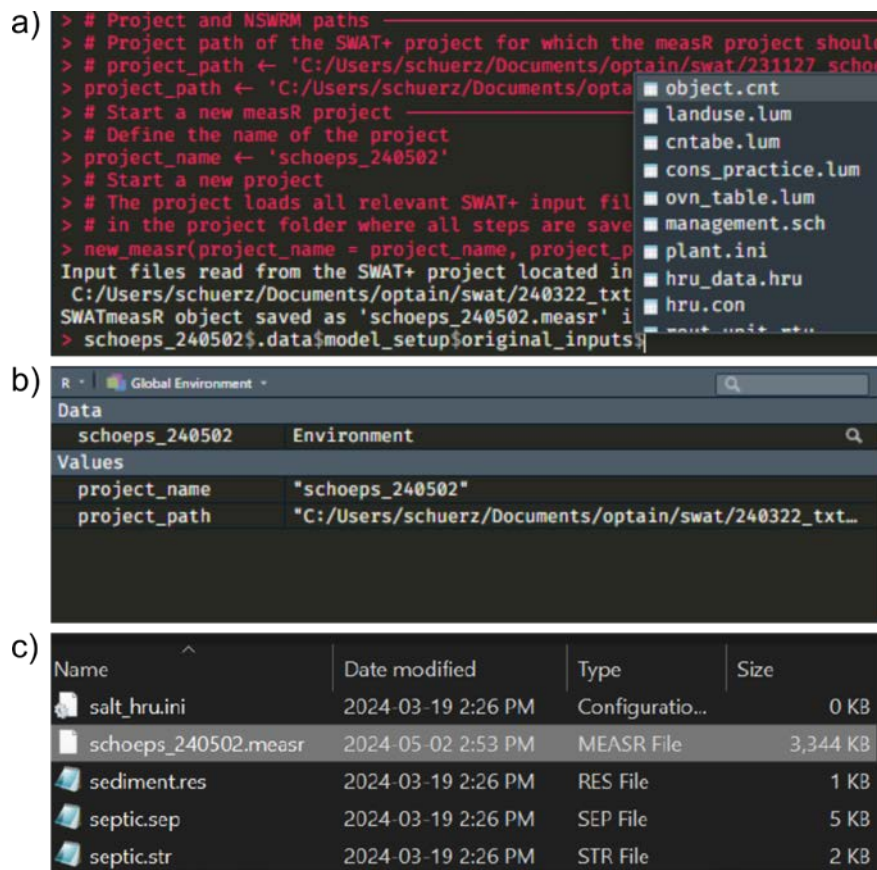


Figure 3.2: The SWATmeasR project with the project_name = 'schoeps_240502', which was generated with the function new_measr()(a). The new measr_project is available as an object in the R environment (b), and as a *.measr file in the SWAT+ project folder (c).

3.2.3. General structure of a SWATmeasR project

The initialised SWATmeasR project is an object in R that can be handled similarly to lists in R. In simple terms, it is an object with several elements that can be accessed with the \$ operator, just like lists in R. The major difference to lists is that the SWATmeasR project also contains functions that affect the internal data elements (functional programming). In the following, the general structure of a SWATmeasR project is outlined. Most of the elements may not be directly relevant for all SWATmeasR users. Nevertheless, it can be helpful when identifying problems or customising the content of a SWATmeasR project if you know where to find what.

Functions and .data

The example in Figure 3.3 shows the initialised project schoeps_240502. The \$ operator can be used to access the elements of schoeps_240502 and all the functions available in the SWATmeasR project become visible:

- `.$reset` resets the SWATmeasR project and the corresponding SWAT+ project folder to its initial condition if NSWRMs were implemented before with `.$implement_nswrm` and `.$write_swat_inputs`.

- `.$save` is usually triggered internally by the other functions and will save the current state of the R object to the `*.measr` file in the SWAT+ project folder. This can be useful if, for example, elements in the SWATmeasR project were adjusted manually.
- `$write_swat_inputs` writes the modified SWAT+ input files into the SWAT+ project folder after NSWORMs were implemented with `.$implement_nswrm`.
- `.$reload_swat_inputs` reloads all SWAT+ input files. This may be necessary if e.g. entries in the SWAT+ input files were missing but are required for the implementation of specific NSWORMs.
- `.$implement_nswrm` implements NSWORMs in the SWAT+ input files in the SWATmeasR object in R (not in the SWAT+ project folder!).
- `.$load_nswrm_location` is used to load the NSWORM locations input file into the SWATmeasR project in R.
- `.$load_nswrm_definition` is used to load an NSWORM definition input file into the SWATmeasR project in R.
- `.$initialize` is the synonymous function to `new_measr()` and should not be called from the SWATmeasR project.
- `.$.__enclos_env__` is some R6 specific argument and is not relevant for the SWATmeasR user.
- `.$data` stores all data of a SWATmeasR project including all SWAT+input tables and the NSWORM definitions.

```

> # Project and NSWORM paths
> # Project path of the SWAT+ project for which the measR proje
> # project_path ← 'C:/Users/schuertz/Documents/optain/swat/233
> project_path ←
> # Start a new
> # Define the n
> project_name ←
> # Start a new
> # The project
> # in the proje
> new_measr(proj
Input files read
C:/Users/schuer
SWATmeasR object
> schoeps_240502

```

Figure 3.3: Overview of the elements of a newly initialised SWATmeasR project.

Meta information:

A SWATmeasR project stores some meta information about the project. The meta information can be accessed as follows:

```

> schoeps_240502$.data$meta

#> $project_name
#> [1] "schoeps_240502"
#>

```

```
#> $project_path
#> [1] "C:/Users/schuerz/Documents/optain/swat/240322_txt"
#>
#> $measr_version
#> [1] "0.8.0"
```

The element `.$data$meta` returns the `project_name`, so the name of the SWATmeasR project in the R environment and on the hard drive, the `project_path`, which is the path from where the SWATmeasR project was loaded (if the project folder was moved before loading the project, this path will also be different), and the version of the SWATmeasR project, which is the R package version of SWATmeasR that was used to build the SWATmeasR project. It is always recommended that the SWATmeasR version of the project and the current version of the R package match.

Model input tables

The SWAT+ model input files, which were read and saved in the SWATmeasR R object, can be found in `.$data$model_setup`. Two copies of the relevant SWAT+ input files are saved in a SWATmeasR project, the unmodified input files stored in `.$data$model_setup$original_inputs` and the input files that will be modified when implementing NSWRMs stored in `.$data$model_setup$modified_inputs`. After the initialisation, the `original_inputs` and the `modified_inputs` are identical. Table 3.1 summarises the SWAT+ input tables that are read and stored in a SWATmeasR project.

Table 3.1: Summary of the SWAT+ input tables that are stored in a SWATmeasR project.

File name	R table name	Definition
object.cnt	object.cnt	Counts of spatial objects in model setup
landuse.lum	landuse.lum	Main land use information file, reference to land use parameter input files
cntable.lum	cntable.lum	Curve numbers for different land use types
cons_practice.lum	cons_practice.lum	USLE P and slope lengths for cons. practices
ovn_table.lum	ovn_table.lum	Overland Manning's n values for different tillage and land cover types
management.sch	management.sch	Management operations schedules file
plant.ini	plant.ini	Plant community input file
hru-data.hru	hru_data.hru	Main HRU input file, which points to HRU parameter input files
hru.con	hru.con	HRU connectivity input file

rout_unit.rtu	rout_unit.rtu	Main routing unit input file, which points to routing unit parameter input files
rout_unit.con	rout_unit.con	Routing unit connectivity input file
rout_unit.def	rout_unit.def	Specification of elements in a routing unit
rout_unit.ele	rout_unit.ele	Lists all elements that are part of a routing unit
chandeg.con	chandeg.con	Channel connectivity input file
reservoir.res	reservoir.res	Main reservoir input file, which points to reservoir parameter input files
hydrology.res	hydrology.res	Reservoir hydrology parameter input file
reservoir.con	reservoir.con	Reservoir connectivity input file
wetland.wet	wetland.wet	Main wetland input file, which points to wetland parameter input files
hydrology.wet	hydrology.wet	Wetland hydrology parameter input file
sediment.res	sediment.res	Reservoir and wetland sediment parameters
nutrients.res	nutrients.res	Reservoir and wetland nutrient parameters
tiledrain.str	tiledrain.str	Tile drainage parameter input file
file.cio	file.cio	Master input file, lists used input files
res_rel.dtl	res_rel.dtl_names	Reservoir release operations file. SWATmeasR project only stores entry names.
lum.dtl	lum.dtl_names	Management decision tables. SWATmeasR project only stores entry names.

Not all of the listed SWAT+ input tables will be modified when NSWORMs are implemented. Some of the input tables are read as lookup tables and to check whether specific entries and parameterisations are defined in a model setup.

In addition to the SWAT+ inputs, `.$data$model_setup$original_inputs` and `.$data$model_setup$modified_inputs` contain elements that track the implementation of NSWORMs in the input tables. The element `file_updated` returns a boolean vector which shows the input files that were updated through the implementation of NSWORMs with `TRUE`.

```
> schoeps_240502$.data$model_setup$modified_inputs$file_updated
#>
#>   object.cnt          file.cio          landuse.lum
#>   FALSE          FALSE          FALSE
#>   cntabe.lum   cons_practice.lum   ovn_table.lum
```

```
#>          FALSE          FALSE          FALSE
#> management.sch          plant.ini          hru_data.hru
#>          FALSE          FALSE          FALSE
#>          hru.con          rout_unit.rtu          rout_unit.con
#>          FALSE          FALSE          FALSE
#> rout_unit.def          rout_unit.ele          chandeg.con
#>          FALSE          FALSE          FALSE
#> reservoir.res          hydrology.res          reservoir.con
#>          FALSE          FALSE          FALSE
#> wetland.wet          hydrology.wet          sediment.res
#>          FALSE          FALSE          FALSE
#> nutrients.res          res_rel.dtl_names          lum.dtl_names
#>          FALSE          FALSE          FALSE
#> tiledrain.str
#>          FALSE
```

The element `files_written` returns whether or not the modified input files were written to the SWAT+ project folder. In the example below for the initial setup of `schoeps_200502`, no input files were written yet.

```
> schoeps_240502$.data$model_setup$modified_inputs$files_written
#>
#> [1] FALSE
```

If NSWORMs were implemented, the element `implemented_nswrms` becomes available in `$.data$model_setup$modified_inputs`. `implemented_nswrms` is a table which tracks the spatial objects in the SWAT+ model setup where NSWORMs were implemented. In the example below from the SWATmeasR project `schoeps_200502`, the NSWORM types `grassslope`, `wetland` and `pond` were implemented in some HRUs. The table `implemented_nswrms` can be accessed as follows:

```
>
schoeps_240502$.data$model_setup$modified_inputs$implemented_nswrms
#>
#> # A tibble: 12 × 5
#>   nswrm      obj_typ obj_id obj_typ_new obj_id_new
#>   <chr>      <chr>  <dbl>      <chr>      <dbl>
#> 1 grassslope hru         1 NA          NA
#> 2 grassslope hru         6 NA          NA
#> 3 grassslope hru         9 NA          NA
#> 4 grassslope hru        12 NA          NA
#> 5 grassslope hru        15 NA          NA
#> 6 grassslope hru         5 NA          NA
#> 7 wetland    hru        238 NA          NA
#> 8 pond       hru        997 res         156
#> 9 pond       hru       1634 res         157
```



```
#> 10 pond      hru      2104 res      155
#> 11 pond      hru      5087 res      158
#> 12 pond      hru      5096 res      159
```

The column `nswrm` indicates the type of NSWORM which was implemented. `obj_type` and `obj_id` define the location where an NSWORM was implemented. NSWORMs are currently only implemented in HRUs, but there may be options to implement NSWORMs in other object types (e.g. channels) in the future. If the implementation of an NSWORM changes the object type of a spatial object, this change is documented in the columns `obj_type_new` and `obj_id_new`. In the case of a pond implementation, the HRU land objects are replaced by reservoir objects. The new object type in this case is therefore `res`. The object ids in the respective SWAT+ input files are specified by `obj_id_new`.

NSWORM definition

If NSWORMs have been defined for a SWATmeasR project, the element `.data$nswrm_definition` becomes available. It collects all relevant input to define the NSWORM parameters and locations of a SWATmeasR project. Most of the elements `.data$nswrm_definition` are explained in more detail in the following sections, as the availability of specific elements depends either on whether certain NSWORMs are defined or whether a definition step has already been carried out or not (e.g. the definition of the NSWORM locations). The only element, which is always available is the `nswrm_lookup`. This provides a general overview of the NSWORMs that were defined for the SWATmeasR project.

```
> schoeps_240502$.data$nswrm_definition$nswrm_lookup
#>
#> # A tibble: 8 × 2
#>   type      nswrm
#>   <chr>     <chr>
#> 1 land_use  buffer
#> 2 land_use  hedge
#> 3 land_use  grassslope
#> 4 land_use  contr_drn
#> 5 management lowtillcc
#> 6 management status_quo
#> 7 wetland  wetland
#> 8 pond     pond
```

`nswrm_lookup` links the general NSWORM types with the actual names of the defined NSWORMs. This is particularly relevant for the NSWORM types `land_use` and `management`, as the example above shows.

3.2.4. NSWRM definition

All NSWRMs that can potentially be implemented in a SWAT+ model setup must be defined. The definition of NSWRMs is done with the function `.$load_nswrm_definition()`. With the current SWATmeasR version 0.8.0, five NSWRM types can be implemented. Table 3.2 summarises the NSWRM types:

Table 3.2: Overview of the NSWRM types that can be implemented with SWATmeasR.

NSWRM type	Definition
management	Farm related measures that cause changes in the management schedules, e.g. conservation farming, reduced tillage, cover crops.
land_use	Any land use change type measure, e.g. implementation of buffer strips, grassed waterways, or afforestation.
wetland	Transformation of a land object to a wetland by adding water storage to the land object and optionally changing land use parameters.
constr_wetland	Implementation of an in stream constructed wetland. In its function this measure is identical to ponds and only different in its naming.
pond	Replacement of a land object by a reservoir object with additional changes in the object routing, e.g. receiving water from channels, routing water directly to channels.

The general way of NSWRM definition is outlined in the small code example below. The required input file, which provides all necessary parametric inputs, is provided with the input argument `file_path`. To indicate which type of NSWRM is defined the correct type must be passed with the argument `type`. The optional argument `overwrite` gives the option to overwrite already existing NSWRM definitions of the same type. By default `overwrite = FALSE`. Therefore, if for example a definition of land use type NSWRMs already exists, but a new one should be loaded, `overwrite` has to be set to `overwrite = TRUE`. The five possible NSWRM types require different definition input files, which will be explained in the following.

```
> land_def_path <- './nswrm_definition/settings_land_use.csv'
> schoeps_240502$load_nswrm_definition(
  file_path = land_def_path,
  type      = 'land_use')
```

Definition of NSWRM `type = 'management'`

The definition of management type NSWRMs requires that the management of the status quo and all potential management scenarios are defined by SWATfarmR projects, which are all located in the SWAT+ project folder. Figure 3.4 shows an example of two `.farm` files of the status quo (`farmR_sq_15yr.farm`) and one scenario (`lowtillcc.farm`). The example shows the `.farm` files only. For retrieving the scheduled operations, also the `.mgts` files for the SWATfarmR projects are required to be

present in the project folder. It is very important at this stage, that the SWATfarmR projects include the current version of the SWAT+ project and no modifications were made to input files which are relevant for management operation scheduling (e.g. *hru-data.hru*, *landuse.lum*, *plant.ini*, or *management.sch*). If any of the files in the SWAT+ project have been updated after the generation of the SWATfarmR projects, those changes will not be present in the SWATfarmR projects and will eventually be missing in the SWATmeasR project.

Name	Date modified	Type	Size
rout_unit.ele	2023-02-24 6:06 PM	ELE File	445 KB
farmR_sq_15yr.farm	2024-02-14 2:27 PM	FARM File	11,895 KB
lowtillcc.farm	2023-10-25 8:21 PM	FARM File	11,905 KB
field.fld	2023-02-24 6:07 PM	FLD File	321 KB
fertilizer.frt	2022-07-26 7:55 PM	FRT File	8 KB

Figure 3.4: SWATfarmR projects located in the SWAT+ project folder.

The input file required to define management type NSWORMs is prepared with the function `prepare_management_scenario_inputs()`, which uses the available SWATfarmR projects. The function performs multiple checks on the provided SWATfarmR projects, such as if all cover the same time period in scheduled operations, or if all schedule names given in the status quo are also available in all scenarios. When processing the management schedules from the SWATfarmR projects, `prepare_management_scenario_inputs()` also compares whether scheduled operations in the status quo are considered to be the same operation in the corresponding schedule in a management. If this is the case, but only the dates between status quo and scenario differ within a time frame of +/- 21 days, the date of the operation in the scenario is set to the date of the corresponding operation in the status quo case. This step ensures that all matching operations are identical and only those operations that are present in a scenario but not in the status quo are different. An example for the implementation of `prepare_management_scenario_inputs()` is given below.

```
> # Path to the SWAT+ project folder
> proj_path <- './txtinout'
> # Name of the status quo SWATfarmR project
> statquo_name <- 'farmR_sq_15yr'
> # If op_data3 labels in status_quo and scenarios are synonymous
> syns <- data.frame(status_quo = c('cultiv25', 'cultiv20'),
                    scenario   = c('fldcull12', 'fldcull12'))
> # Path to write the management definition file
> mgt_wrt_path <- './nswrm_definitions'
>
> prepare_management_scenario_inputs(project_path = proj_path,
                                   status_quo   = statquo_name,
                                   synonyms     = syns,
                                   write_path   = mgt_wrt_path)
```

The input argument `project_path` specifies the path to the SWAT+ project folder. To identify the SWATfarmR project, which has to be considered as the status quo project, the argument `status_quo` must provide the name of the SWATfarmR project. `synonyms` is an optional input argument to define operations that the date correction routine should consider as identical operations, even though the `op_data3` labels of the operation in the status quo and in a scenario are different. This may be the case, for example, if the tillage operation in the status quo and a conservation tillage scenario should be on the same day, but the tillage type has changed in the conservation tillage scenario. The `write_path` defines where the management type definition file should be written to.

Once all checks on the SWATfarmR projects and the date correction of the scenario operation schedules have been successful, the management type definition file is written to the defined path. The returned file is an `.rds` file, which always has the timestamp of its generation as a prefix, followed by the name 'mgt_scenarios'. Figure 3.5 shows an example of a successfully written management input file which can then be used to load the management type NSWRM definition.

Name	Date modified	Type	Size
20231128_1751_mgt_scenarios	2023-11-30 2:14 PM	File folder	
20231128_1751_mgt_scenarios.rds	2023-11-28 5:51 PM	RDS File	1,079 KB
measure_location.csv	2023-11-02 4:53 PM	Microsoft E...	12 KB
settings_land_use.csv	2023-11-30 1:52 PM	Microsoft E...	1 KB
settings_pond.csv	2024-05-02 1:29 PM	Microsoft E...	1 KB

Figure 3.5: Example of a management input file, which was generated with the function `prepare_management_scenario_inputs()`.

The generated management input file can then be loaded into the SWATmeasR project with the function `$.load_nswrm_definition()`. As shown in the example below the `file_path` must be the one of the generated management `.rds` file and `type` must be 'management'.

```
> mgt_def_path <-
  './nswrm_definition/20231128_1751_mgt_scenario.rds'
> schoeps_240502$.load_nswrm_definition(
  file_path = mgt_def_path,
  type      = 'management')
```

After loading the management definition file, the management NSWORMs are added to the lookup table `$.data$nswrm_definition$nswrm_lookup`, and the element management is added to `$.data$nswrm_definition` which stores the management inputs for the status quo and for all scenario operation schedules. As for the implementation of management operations not only the management schedules in `management.sch` are relevant, but also the corresponding plant communities and land use definitions, all management scenarios and the status

quo provide the input files `hru_data.hru`, `landuse.lum`, `management.sch`, and `plant.ini`.

Definition of NSWRM type = 'land_use'

The definition of land use type NSWORMs requires the definition of their general land use information into which the land use of a land object will be converted when the NSWORM is implemented. The land use definitions are prepared in a *.csv file and loaded into the SWATmeasR project with the function `.$load_nswrm_definition()`. An example land use input table is shown in Table 3.3.

Table 3.3: Example land use input table for loading land use type NSWORMs into a SWATmeasR project.

nswrm	plnt_com	mgt	cn2	cons_prac	ov_mann	tile	lum_dtl
buffer	rnge_test_com	rnge_mgt	pasth	greening	densegrass	null	
hedge	frsd_com	hedge_mgt	wood_g	greening	forest_light		
grassslope	rnge_test_com	rnge_mgt	pasth	greening	densegrass		
contr_drn						mw24_1000	control_drainage

The example in Table 3.3 defines the four land use type NSWORMs `buffer`, `hedge`, `grassslope` and `contr_drn` (controlled drainage). The labels defined in the column `nswrm` will be the names of the measures in the SWATmeasR project. The columns `plnt_com` to `tile` are the entries in the `landuse.lum` SWAT+ input file which are written when a defined land use type measure is implemented in the SWAT+ model setup. The labels in the columns are references to other SWAT+ input files, whereby, for example, the label in the `plnt_com` column refers to a plant community in the `plant.ini` input file or `mgt` refers to a management schedule in the `management.sch` file. The last column `lum_dtl` points to entries in the `lum.dtl` decision table. Each of the entries must already be defined in the respective input files when the land use type NSWORM definition is loaded. `.$load_nswrm_definition()` checks that all of the entries are already defined for the SWAT+ model setup and returns an error if some of the land use definitions are missing.

The example in Figure 3.6 shows the error that is returned if entries are missing in the SWAT+ input files. The error message shows that the entry `'wrong_com'` is not defined in `plant.ini` and `'missing_mgt'` is not defined in `management.sch`. The error message provides guidance on how to proceed. To resolve the problem, the user must add the missing entries in the respective SWAT+ input files. As mentioned above, SWATmeasR takes a 'snapshot' of the SWAT+ input files when a new SWATmeasR project is generated. To update the input tables in the SWATmeasR project, all input files from the project folder must therefore be reloaded into the SWATmeasR project using the function `.$reload_swat_inputs()`. The user can then proceed loading the NSWORM definition file for land use.

```

> schoeps_240502$load_nswrm_definition(luse_path, 'land_use', overwrite = T)
Error in load_luse_def(file_path, swat_inputs) :
  The following options are not defined in the respective SWAT+ input files:

'plnt_com' not defined in 'plant.ini': wrong_com
'mgt' not defined in 'management.sch': missing_mgt

Please do the following to solve this issue:
i) Add the missing entries in the SWAT+ input files
ii) Reload all SWAT+ input files with measr_project$reload_swat_inputs()
    (can only be done when no NSWORMs wer implemented yet)
iii) Load again 'land_use' definition table with measr_project$load_nswrm_definition()

```

Figure 3.6: Error message when entries in SWAT+ input files are missing which were defined in the land use type NSWORM definition file.

The example in Table 3.3 shows empty elements, for example all *landuse.lum* entries for *contr_drn*. An empty element means that the initial value of this element remains unchanged when implementing a land use NSWORM in an HRU. In the case of *contr_drn* this would mean that all *landuse.lum* entries remain unchanged and only a controlled drainage is added to the HRU. If a *landuse.lum* entry should be specifically deactivated with the implementation of a land use type NSWORM, this can be done with the label 'null'. This is shown in Table 3.3 for the definition of 'buffer' where *tile* should be set to 'null' in all cases where buffer strips are implemented.

Once the land use type NSWORM input csv file has been generated, it can be loaded into the SWATmeasR project as shown below for the example SWATmeasR project *schoeps_240502*:

```

> land_def_path <- './nswrm_definition/settings_land_use.csv'
> schoeps_240502$load_nswrm_definition(
  file_path = land_def_path,
  type      = 'land_use')

```

Definition of NSWORM type = 'wetland'

The definition of wetlands is prepared in a *.csv input file. Table 3.4 provides an example for a wetland input table. The definition of a wetland at least requires to define the ID of the HRU (*hru_id*) to which a wetland water storage should be added. Further, a *lu_mgt* entry can be defined if the wetland should have a different land use than the initial one when the wetland is implemented. By default, the connectivity of the land object where a wetland is implemented remains unchanged, and water and nutrient fluxes are routed to the neighbouring objects as initially defined. If instead water and nutrient fluxes should be routed directly to a channel after the implementation of a wetland, the channel to which fluxes should be routed can be defined with *cha_to_id*. The columns *hru_ps* to *hru_frac* correspond to the parameter entries in the SWAT+ input file *hydrology.wet*, while the columns *rel* to *nut* correspond to the columns with the same names in the file *wetland.wet*, which hold pointer labels to entries in other SWAT+ input files.

Table 3.4: Example wetland input table for loading wetland type NSWORMs into a SWATmeasR project.

hru_id	lu_mgt	cha_to_id	hru_ps	dp_ps	hru_es	dp_es	k	evap	vol_area_co	vol_dp_a	vol_dp_b	hru_frac	rel	sed	nut
238	rnge_lum												wetland	sedwet1	nutwet1
5096		524													

All columns except the `hru_id` are optional inputs and default values are implemented if an entry is kept empty or if the column is not provided at all. Table 3.5 provides an overview of the optional inputs together with the default values to which the parameters are set if they were not provided in the wetland definition input file.

Table 3.5: Default values and definitions of wetland parameters that can be defined in the wetland NSWORM input file.

Parameter	Default value	Definition
<code>lu_mgt</code>	keep initial*	Name of landuse definition in <code>landuse.lum</code> .
<code>cha_to_id</code>	keep initial**	Channel ID to which water and nutrient fluxes are routed.
<code>hru_ps</code>	0.1	Fraction of HRU area at principal spillway.
<code>dp_ps</code>	20.0	Average depth of water at principal spillway in mm.
<code>hru_es</code>	0.25	Fraction of HRU area at emergency spillway.
<code>dp_es</code>	100.0	Average depth of water at emergency spillway in mm.
<code>k</code>	0.01	Hydraulic conductivity of the wetland bottom in mm/hr.
<code>evap</code>	0.7	Wetland evaporation coefficient.
<code>vol_area_co</code>	1.0	Volume surface area coefficient for HRU impoundment.
<code>vol_dp_a</code>	1.0	Volume depth coefficient a for HRU impoundment.
<code>vol_dp_b</code>	1.0	Volume depth coefficient b for HRU impoundment.
<code>hru_frac</code>	0.5	Fraction of HRU that drains into wetland.
<code>rel</code>	'wetland'***	Pointer to the reservoir and wetland release decision table.
<code>sed</code>	'sedwet1'***	Pointer to the reservoir and wetland sediment file.
<code>nut</code>	'nutwet1'***	Pointer to the reservoir and wetland nutrient file.

* keep the initial `landuse.lum` entry for an HRU, ** keep the initial routing of the HRU, *** set those default values if provided in the respective input files, otherwise set 'null'.

Once the wetland input csv file is generated, it can be loaded into the SWATmeasR project as shown below for the example SWATmeasR project `schoeps_240502`:

```
> wetl_def_path <- './nswrm_definition/settings_wetland.csv'
```

```
> schoeps_240502$load_nswrm_definition(
  file_path = wetl_def_path,
  type      = 'wetland')
```

Definition of NSWRM type = 'pond' and type = 'constr_wetland'

The definitions of ponds and constructed wetlands (*constr_wetland*) are prepared in *.csv input files. Although the implementation of these two NSWRM types in a SWAT+ model setup is identical, their definition is done in separate input files that have the same structure. Table 3.6 provides an example for such an input table. The definition of a pond or *constr_wetland* requires at least the definition of the columns *hru_id* and *cha_to_id*. In contrast to wetlands, *hru_id* can include multiple HRU IDs which are merged into one reservoir object when the NSWRM is implemented. *cha_to_id* must always be a single channel ID to which the fluxes of the new reservoir object are routed. In addition, a *cha_from_id* can be defined if the new reservoir object should also receive fluxes from channel objects. In this case also the routing of the channel objects which were defined with *cha_from_id* is changed to route into the new reservoir object. This can for instance be useful if a pond or constructed wetland is integrated into the existing channel network. The columns *area_ps* to *shp_co2* correspond to the parameter entries in the SWAT+ input file *hydrology.res*, while the columns *rel* to *nut* correspond to the columns with the same names in the file *reservoir.res*, which hold pointer labels to entries in other SWAT+ input files.

Table 3.6: Example pond or *constr_wetland* input table for loading pond or *constr_wetland* type NSWRM into a SWATmeasR project.

<i>hru_id</i>	<i>cha_to_id</i>	<i>cha_from_id</i>	<i>area_ps</i>	<i>vol_ps</i>	<i>area_es</i>	<i>vol_es</i>	<i>k</i>	<i>evap_co</i>	<i>shp_co1</i>	<i>shp_co2</i>	<i>rel</i>	<i>sed</i>	<i>nut</i>
153, 154, 157	475		0.173	0.253	0.211	0.604					dd_days2		
201, 234	565		0.126	0.185	0.154	0.441					dd_days2		
997	846		0.070	0.140	0.087	0.262							
1634	941		0.052	0.103	0.065	0.194							
2104	613		0.000	0.000	0.000	0.000							
5087	395	604, 605	0.052	0.104	0.065	0.194							
5096	934	933	0.084	0.168	0.105	0.316							

The *cha_from_id*, the *hydrology.res*, and the *reservoir.res* parameters are optional inputs and default values are implemented if an entry is kept empty or if a column is not provided at all. Table 3.7 provides an overview of the optional inputs together with the default values to which the parameters are set if they were not provided in the wetland definition input file. If the user provides values for the areas and volumes of the new reservoir object, the function `$load_nswrm_definition()` performs checks for the provided values. The provided areas *area_ps*, and *area_es* must be less than the sum of the areas of the replaced HRUs, as the new water surface cannot be larger than the replaced land surfaces. Further, *area_ps* must be less than or equal to the *area_es* and the *vol_ps* must be less than or equal to the *vol_es*, as the values at the emergency spillway water level cannot be less than they would be at principal spillway water levels.

Table 3.7: Default values and definitions of reservoir parameters which can be defined in the pond and constr_wetland NSWRM input files.

Parameter	Default value	Definition
cha_from_id	no change*	Channel ID from which water and nutrient fluxes are routed into the new reservoir object.
area_ps	0.1	Reservoir surface area at principal spillway water level in ha.
vol_ps	20.0	Reservoir volume at principal spillway water level in ha-m***.
area_es	0.25	Reservoir surface area at emergency spillway water level in ha.
vol_es	100.0	Reservoir volume at emergency spillway water level in ha-m***.
k	0.01	Hydraulic conductivity of the reservoir bottom in mm/hr.
evap_co	0.7	Reservoir evaporation coefficient.
shp_co1	1.0	Shape coefficient 1 for reservoirs.
shp_co2	1.0	Shape coefficient 2 for reservoirs.
rel	'wetland'**	Pointer to the reservoir and wetland release decision table.
sed	'sedwetl'**	Pointer to the reservoir and wetland sediment file.
nut	'nutwetl'**	Pointer to the reservoir and wetland nutrient file.

* keep the initial routing of the channel objects, ** set those default values if provided in the respective input files, otherwise set 'null'. *** ha-m (hectare * metres) is equivalent to 10,000 m³.

Once the pond or constr_wetland input csv file has been generated, it can be loaded into the SWATmeasR project as shown below for the example SWATmeasR project schoeps_240502:

```
> # In case of ponds
> pond_def_path <- './nswrm_definition/settings_pond.csv'
> schoeps_240502$load_nswrm_definition(
  file_path = pond_def_path,
  type      = 'pond')
> #
> # In case of constructed wetlands
> cwtl_def_path <- './nswrm_definition/settings_constr_wetl.csv'
> schoeps_240502$load_nswrm_definition(
  file_path = cwtl_def_path,
  type      = 'constr_wetland')
```

3.2.5. Definition of NSWRM locations

After all NSWRMs that can be implemented in a SWAT+ model setup were defined and loaded into the SWATmeasR project, the user must define the potential locations of the NSWRMs. The definition of the NSWRM locations is done in a *.csv locations input file and loaded into the SWATmeasR project with the function `$.load_nswrm_location()`. Table 3.8 shows an example of an NSWRM locations input table. The input table requires the four columns `id`, `name`, `nswrm`, and `obj_id`. `id` defines the ID of an NSWRM which is later used as an identifier for the implementation of NSWRMs. `name` is the unique name of a measure location and can be defined freely by the user. This column is not used at the moment. `nswrm` refers to the entries in the definition files for the respective NSWRM types. The labels of the specific NSWRMs must be used here to identify the respective measures, for example `buffer`, `hedge`, or `grassslope` in the example of the SWATmeasR project `schoeps_250502` when referring to the different land use type NSWRMs. `obj_id` refers to the IDs of the spatial objects (HRU IDs) in which an NSWRM is implemented. For management and land use type NSWRMs, multiple HRU IDs can be grouped together in one `obj_id` entry to define a single NSWRM location. When such an NSWRM is implemented, the change in management or land use is implemented in all of the defined HRUs at the same time. For wetland, pond and `constr_wetland` NSWRMs the `obj_id` must match the respective entry in the column `hru_id` in the wetland or pond (`constr_wetland`) definition file.

Table 3.8: Example for a NSWRM location definition table for loading measure locations into a SWATmeasR project.

id	name	nswrm	obj_id
1	buffer_1	buffer	479
2	buffer_2	buffer	710
3	buffer_3	buffer	468
4	buffer_4	buffer	337
5	buffer_5	buffer	206, 287, 856
6	buffer_6	buffer	140, 142
7	grassslope_1	grassslope	201, 203
8	grassslope_2	grassslope	122
9	grassslope_3	grassslope	607, 608, 610, 613
10	hedge_1	hedge	5278, 5284
11	hedge_2	hedge	126
12	hedge_3	hedge	122, 123, 124
13	hedge_4	hedge	74, 75, 76, 83
14	lowtillcc_1	lowtillcc	1, 2, 5, 9, 10, 11
15	lowtillcc_2	lowtillcc	17, 18
16	lowtillcc_3	lowtillcc	19, 20, 21, 22, 23
17	lowtillcc_4	lowtillcc	25
18	lowtillcc_5	lowtillcc	49, 50, 51, 52
19	pond_1	pond	468
20	pond_2	pond	5087

Once the NSWRM location csv file is generated, it can be loaded into the SWATmeasR project as shown below for the example SWATmeasR project `schoeps_240502`. Similar to loading NSWRM definitions, the optional argument `overwrite` is required in `.$load_nswrm_location()` to overwrite an existing locations table if necessary.

```
> location_path <- './nswrm_definition/nswrm_location.csv'  
> schoeps_240502$load_nswrm_location(file_path = location_path)
```

3.2.6. NSWRM implementation

Although the implementation of NSWRMs is not part of the generation of a SWATmeasR project, it is briefly outlined here. The NSWRM implementation is part of the SWAT+ workflow in the optimization routine, and the function calls to implement NSWRMs and to write modified SWAT+ input tables into the SWAT+ project folder will be executed in the R script `SWAT.R` (see also section 4.3.2). It is, however, good practice to perform tests with the NSWRM implementation, run SWAT simulations and analyse the impacts of the implemented NSWRMs for plausibility.

Once all NSWRMs and their potential locations were defined, they can be implemented in the corresponding SWAT+ model setup. The NSWRM implementation is always a two-stage procedure. First, the measures are implemented in the SWAT+ input tables stored in the SWATmeasR project in `.data$model_setup$modified_inputs` using the function `.$implement_nswrm()`. In a second step, the changed SWAT+ input files which were modified in the SWATmeasR project are written into the SWAT+ project folder with the function `.$write_swat_inputs()` and overwrite the initial input files.

The NSWRM locations are implemented by providing their `id` value with the argument `nswrm_id` in the function `.$implement_nswrm()`. `nswrm_id` can be a single value if only one NSWRM location should be implemented, but can also be a vector of IDs if multiple locations should be implemented. In the code example below, the NSWRM locations of `buffer_1` to `buffer_4`, `grassslope_2`, `hedge_3`, and `pond_1` should be implemented, which were defined in the example in Table 3.8. From the table we can read the IDs of those NSWRM locations, which are `nswrm_id = c(1:4, 8, 12, 19)`.

```
> schoeps_240502$implement_nswrm(nswrm_id = c(1:4, 8, 12, 19))
```

The implementation of NSWRMs can be verified by looking into the table `implemented_nswrms`, which was generated when the measures were implemented. The table can be accessed as shown below:

```

>
schoeps_240502$.data$model_setup$modified_inputs$implemented_nswrm
s
#>
#> # A tibble: 7 × 5
#>   nswrm  obj_typ obj_id obj_typ_new obj_id_new
#>   <chr>  <chr>     <dbl> <chr>         <dbl>
#> 1 buffer hru         479   NA             NA
#> 2 buffer hru         710   NA             NA
#> 3 buffer hru         337   NA             NA
#> 4 hedge  hru         122   NA             NA
#> 5 hedge  hru         123   NA             NA
#> 6 hedge  hru         124   NA             NA
#> 7 pond   hru         468   res            156

```

The table shows all HRUs where NSWORMs have been implemented. A closer look shows that the implementations of grassslope_2 (in HRU 122) and buffer_3 (in HRU 468) are missing. The reason for that is an overlap with the implementation of other NSWORMs in the same HRUs. HRU 122 was also affected by the implementation of hedge_3 and HRU 468 eventually was replaced by a pond, which is represented by the reservoir 156. As the example shows, there is a specific order in which NSWORMs are implemented between the different NSWORM types, but also within the same type. Figure 3.7 shows the order in which the NSWORMs are implemented. The NSWORM types are processed in the sequence management > land_use > wetland > constr_wetland > pond. Within each NSWORM type, the measures are implemented in the order in which they were provided in the NSWORM location table.

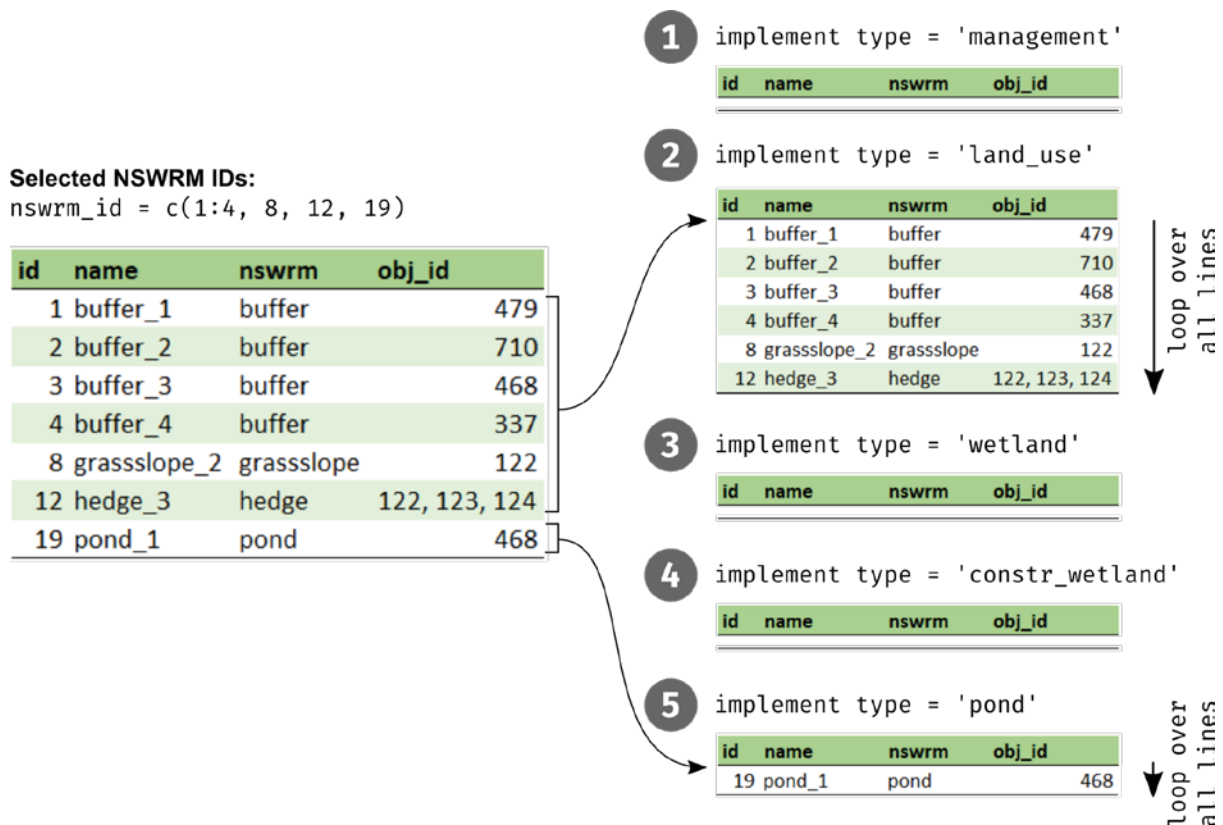


Figure 3.7: Order of implementation of NSWRMs.

In the illustrated example, hedge_3 is implemented in HRU 122 after the implementation of grassslope_2 in the same HRU and therefore overrules the first implementation. pond_1 is implemented as the last measure in the shown sequence. It replaces the HRU 468 with a new reservoir and therefore overrules the implementation of buffer_3 in the same HRU.

Once the NSWRM locations were implemented, the changes in the SWAT+ model input files can be written into the SWAT+ project folder. This is done as follows:

```
> schoeps_240502$write_swat_inputs()
```

Once implemented, SWAT simulations can be executed and the simulated outputs can be analysed to assess the impacts of the implemented NSWRMs. After finishing the simulation, the original SWAT+ model setup can be restored with the function `.$reset()`. With the reset all modified input files are overwritten by the original input files stored in `.$data$model_setup$original_inputs`. It is strongly advised to always reset the SWAT+ project setup after the simulation and analysis of results is completed. Otherwise, there may be the risk that NSWRMs will remain in the model input files and the generation of a new SWATmeasR project for example would read the modified input files and would therefore start with a wrong initial model setup.

```
> schoeps_240502$reset()  
#>  
#> Resetting input files in project folder... Done!  
#> Resetting tables measR project ... Done!
```

4. Running the optimisation

4.1. python installation

CoMOLA is written in the script language `python`. Therefore, `python` must be installed on the computer in order to run CoMOLA, specifically `python` version 3.11.x. There are different approaches to install `python`, and it is very likely that a version (or several versions) of `python` are already installed on the computer on which CoMOLA is to be run. To minimise the risk of any CoMOLA/`python` related issues, a clean new installation of `python` and required `python` packages is recommended. The installation of `python` and required packages will be done using `miniconda`. `miniconda` is a lightweight `python` package manager (a minimum version of the well-known package manager `anaconda`) that allows users to easily install different versions of `python` and required `python` packages.

4.1.1. miniconda installation

Two approaches to installing `miniconda` are shown below, one that runs the installation automatically via Windows `powershell` and the second that uses the Windows installer. If it is OK to install the software in the computer's default programs folder, simply the command line approach can be used.

Installation from the command line

To run the installation from the command line, the Windows `powershell` must be started. It is started by clicking on the Windows Start button, typing "cmd" and pressing Enter (see Figure 4.1).

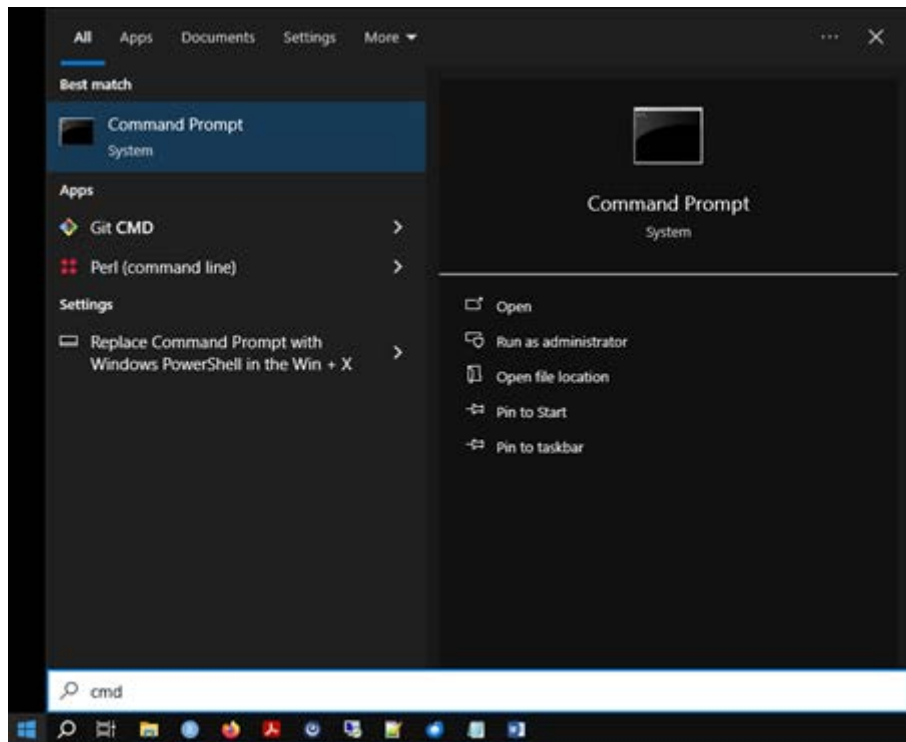


Figure 4.1: Starting the Windows command prompt (`powershell`) from the Windows start menu.

A command prompt opens after pressing enter. The `miniconda` installation is executed in the command prompt by running the lines of code from the code box below. Just copy the few lines of code into the command line. The commands will execute the latest version of the `miniconda` installer. It downloads the executable file, installs it in the default program path and deletes the installer executable file after the installation.

```
curl https://repo.anaconda.com/miniconda/Miniconda3-latest-Windows-x86_64.exe -o miniconda.exe
start /wait "" miniconda.exe /S
del miniconda.exe
```

Installation with the installer

Alternatively, the installer file can also be downloaded manually from the anaconda website. The latest Windows installer executable file can be retrieved from <https://docs.anaconda.com/free/miniconda/>. After downloading the installer, `miniconda` can be installed to the desired location on the computer's hard drive.

4.1.2. miniconda setup

The `miniconda` installation includes the installation of an own prompt, which is different to the Windows command prompt (`cmd`). To avoid registering the paths of `conda` and `python` in the Windows PATHs (which may cause troubles with other software) it is safer to work directly in the base environment of `miniconda`. The `miniconda` prompt can be started in the same way as the Windows command prompt, by clicking on the Windows start button but searching for the “Anaconda Prompt (`miniconda3`)” (see Figure 4.2).

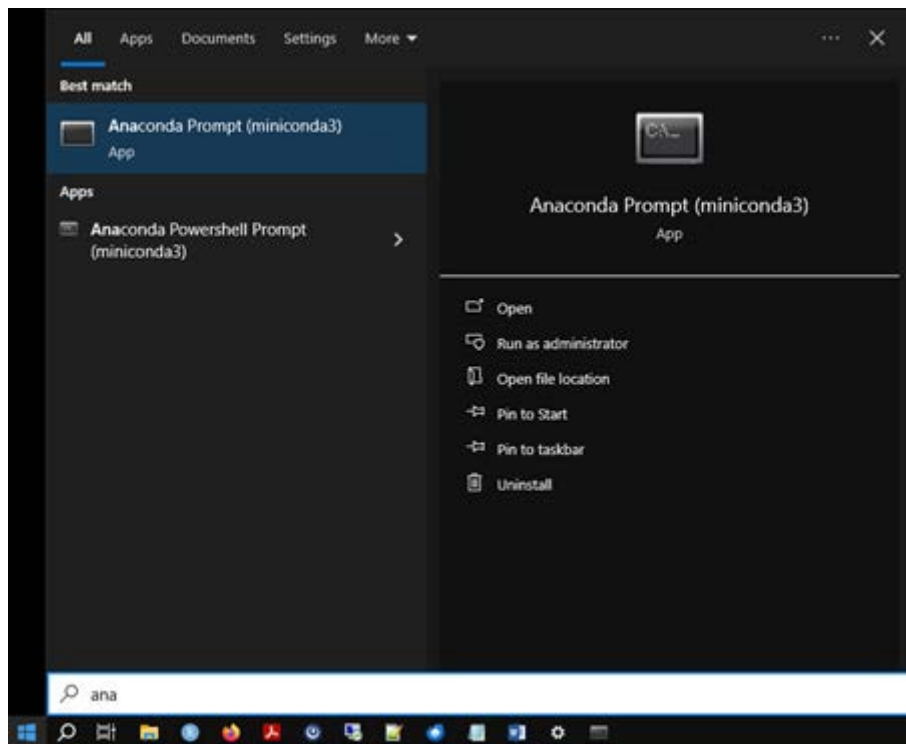


Figure 4.2: Starting the Anaconda Prompt (`miniconda3`) from the Windows start menu.

`python` installation and installation of required `python` packages

By starting the Anaconda prompt, a new command window opens which should look similar as below (Figure 4.3). In brackets the cursor position shows `(base)` before the home path, which indicates that the user is currently working in the `miniconda` base environment.

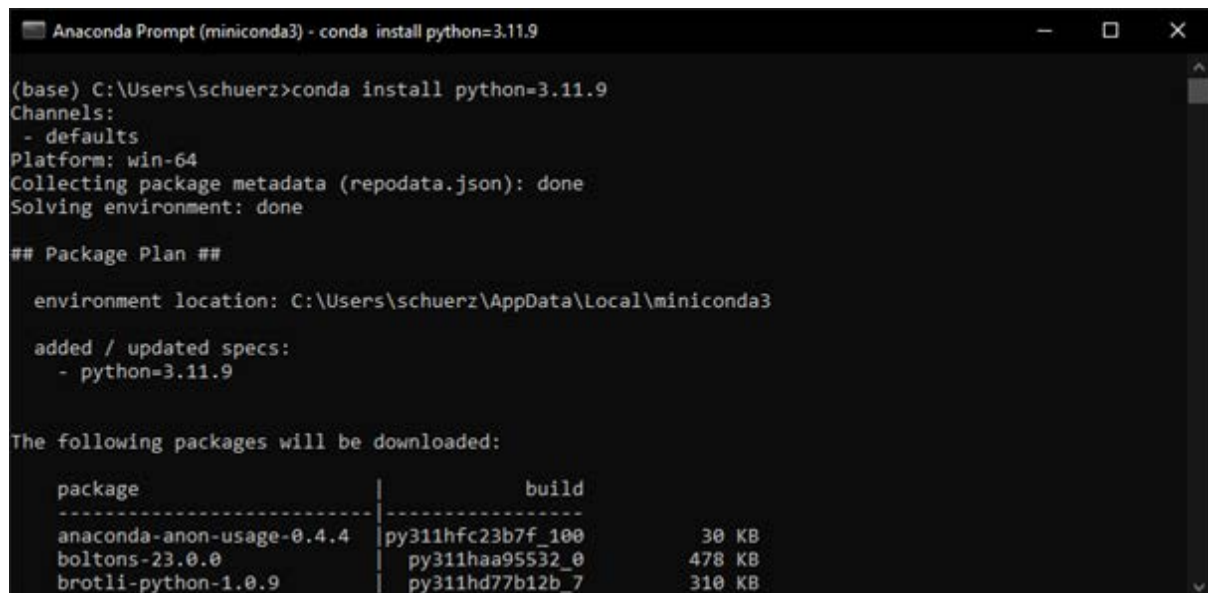


Figure 4.3: The Anaconda Prompt (`miniconda3`) after starting.

Due to the requirements of CoMOLA, the newest version of python 3.11x needs to be installed, which at the time of writing this document was version 3.11.9. To install specifically this `python` version the command below is executed in the Anaconda prompt:

```
conda install python=3.11.9
```

Executing the install command starts the `python` 3.11.9 installation, which should look similar to the screenshot below (Figure 4.4).



```

Anaconda Prompt (miniconda3) - conda install python=3.11.9
(base) C:\Users\schuerz>conda install python=3.11.9
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\schuerz\AppData\Local\miniconda3

added / updated specs:
 - python=3.11.9

The following packages will be downloaded:

package | build | size
-----|-----|-----
anaconda-anon-usage-0.4.4 | py311hfc23b7f_100 | 30 KB
boltons-23.0.0 | py311haa95532_0 | 478 KB
brotli-python-1.0.9 | py311hd77b12b_7 | 310 KB
  
```

Figure 4.4: The `python` installation process in the Anaconda Prompt (miniconda3).

During the installation process, you will be prompted to downgrade all previously installed `python` packages to the version 3.11 (which is not the latest `python` version). Pressing enter will continue the installation process.

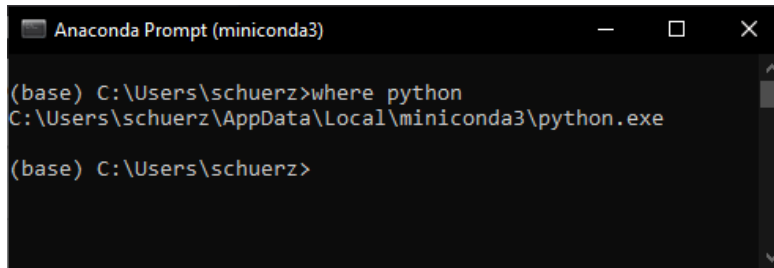
CoMOLA depends on the `python` package `numpy`. `numpy` is not installed by default with the `miniconda` installation. `python` packages can be installed with the command `conda install`. The `numpy` installation is done by executing the code in the code box below in the Anaconda prompt. Again, the installation process will ask to proceed. Pressing enter continues the installation process.

```
conda install numpy
```

Checking the `python` and `numpy` installation

To start CoMOLA, the path to the correct `python` executable file (with the version 3.11.x) must be specified in the input file `config.ini` (see section 4.2). The path to the current `python` executable can be found with the command `where` in the Anaconda prompt (see example in Figure 4.5).

```
where python
```



```

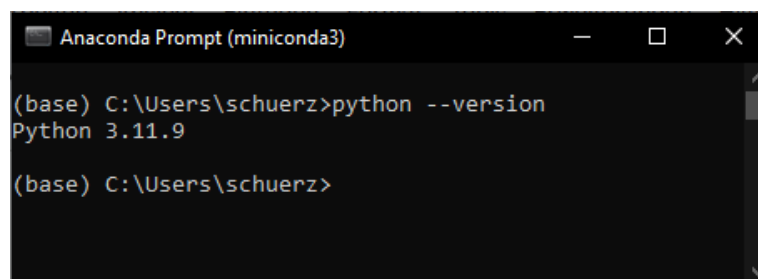
Anaconda Prompt (miniconda3)
(base) C:\Users\schuerz>where python
C:\Users\schuerz\AppData\Local\miniconda3\python.exe
(base) C:\Users\schuerz>

```

Figure 4.5: The `python` executable path is returned with the command `where`.

The `python` version can be checked as follows. If the installation was successful the `python` version 3.11.9 is shown (example in Figure 4.6).

```
python --version
```



```

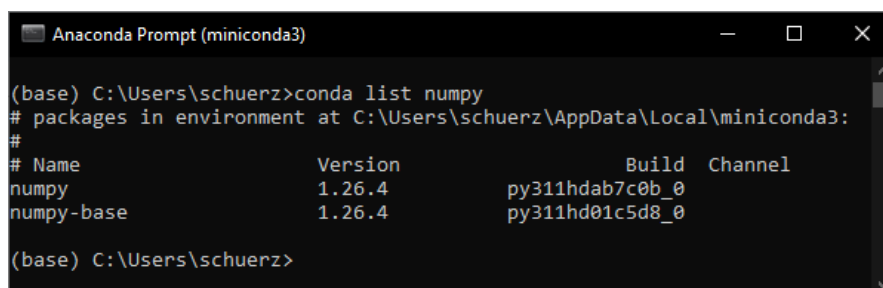
Anaconda Prompt (miniconda3)
(base) C:\Users\schuerz>python --version
Python 3.11.9
(base) C:\Users\schuerz>

```

Figure 4.6: Checking the version of the `python` executable.

Checking if `numpy` is installed can be done with the command `conda list`. The checking for `numpy` is performed as follows. A list of `numpy` related packages should be returned (see example in Figure 4.7).

```
conda list numpy
```



```

Anaconda Prompt (miniconda3)
(base) C:\Users\schuerz>conda list numpy
# packages in environment at C:\Users\schuerz\AppData\Local\miniconda3:
#
# Name          Version      Build          Channel
numpy           1.26.4       py311hdab7c0b_0
numpy-base     1.26.4       py311hd01c5d8_0
(base) C:\Users\schuerz>

```

Figure 4.7: Listing all `numpy` packages with the command `conda list`.

4.2. CoMOLA file structure

After successfully installing `python`, the CoMOLA optimisation software is to be downloaded from GitHub (https://github.com/michstrauch/CoMOLA_SWATplus) or the UFZ GitLab: <https://git.ufz.de/optain/wp5-optimisation/comola>. This section briefly describes the different folders and files of CoMOLA. Section 4.3 provides a detailed manual on how to set up and run the optimisation for a user-specific case study. The most important rule when using CoMOLA is not to change its file structure, including all file names (Figure 4.8).

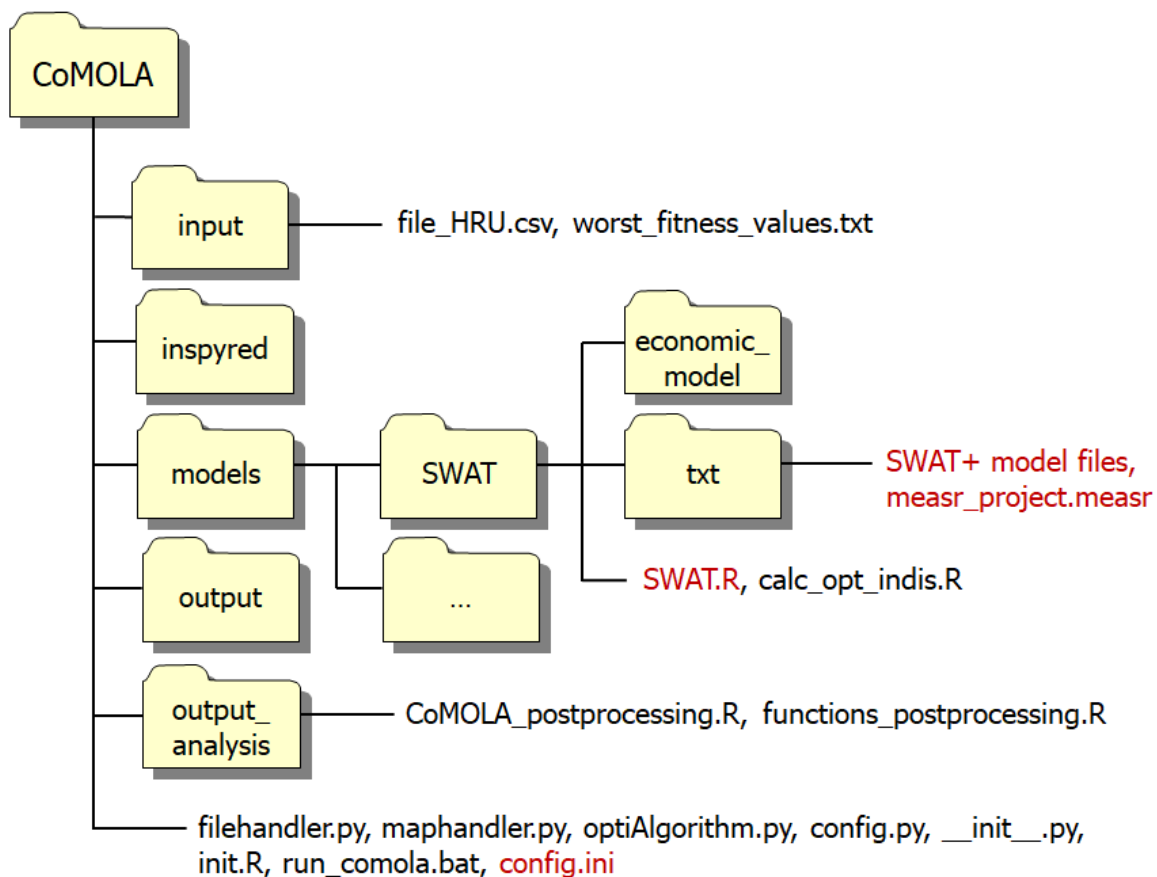


Figure 4.8: CoMOLA file structure. Files in red fonts need to be added, replaced or adjusted by case studies.

The folder *input* contains two files that are automatically written when the optimisation is started. Users should not edit these files. *file_HRU.csv* specifies a vector of NSWRM sites to be included in the optimisation and the state of implementation at the start of the optimisation process. The vector represents the genome of the starting individual (i.e. individual 1 in the initial population) and by default contains the value of 1 for all genes (i.e. no NSWRM are implemented). The file *worst_fitness_values.txt* defines the fitness values to be used for infeasible individuals (individuals that violate certain constraints). Since we do not define any optimisation constraints in OPTAIN, this file is actually obsolete. However, the current version of CoMOLA requires such a file.

The *inspyred* folder contains modules from the *inspyred* python package (Garret, 2012), including optimisation algorithms such as NSGA-II (Section 2.4). Users should not deal with this folder.

The folder *models* must contain the SWAT+ model files and, if required, any other model that needs to be included for calculating the (fitness) values of the optimisation objectives. In OPTAIN, it is sufficient to include the SWAT+ model only. Users need to add their own SWAT+ project (*txt*) in a folder called *SWAT* and make sure that a number of model requirements is met (further details in section 4.3). The folder *economic_model* contains the R scripts and input tables required to calculate the socio-economic performance indicators. At the time of writing this document, the files were still under preparation (Deliverable D4.5). They will be added as soon as possible. In addition, the *models* folder contains two R files (by default). The *calc_opt_indis.R* file defines all available functions for calculating the environmental performance indicators. If additional functions are required to calculate the optimisation objectives, the user must define them here or send a request to the authors of this report. The *SWAT.R* file must be adapted to specify the objectives used for the optimisation (see section 4.3).

The *output* folder is the directory where the log file (**_optimisation_log.txt*) and fitness values for each individual NSWRM plan tested in the optimisation (**_individuals_file.csv*) are printed. Genomes and fitness values of the final Pareto optimal solutions are printed in the log file. The *output* file names contain a timestamp (*dd_MM_yyyy_hh_mm_ss*) which indicate the time at which the respective optimisation run was started. Users should access the results by running the *CoMOLA_postprocessing.R* file (section 4.4), which is provided in the *output_analysis* folder along with a file containing the postprocessing functions (*functions_postprocessing.R*).

In addition to the folders already presented, the *CoMOLA* folder contains the *init.R* file, which automatically writes the required files to the *input* folder, as well as a number of python files needed to run the optimisation (users should not bother with these). The only file in the main folder that is relevant for the user is the CoMOLA master file (*config.ini*). This file must be adapted by defining the paths to the R and python executables, as well as important optimisation parameters such as population size and maximum number of generations (see Section 4.3).

4.3. Setting up and starting an optimisation run

At this point in the protocol we assume that the user has installed python, downloaded CoMOLA (<https://git.ufz.de/optain/wp5-optimisation/comola>) and familiarised him or herself with the file structure of the optimisation tool (section 4.1 and 4.2). To set up an optimization run, three main steps must be taken:

- (1) Add your own *txt* folder
- (2) Adjust the *SWAT.R* script for your optimisation objectives
- (3) Configure the master file (*config.ini*)

4.3.1. Add your own txt folder

Create a copy of your SWAT+ project (txt folder) and add it to the *models/SWAT* directory. Note that the SWAT+ project folder must be named 'txt'. In the *txt* folder, make sure:

1. That the simulation period is correctly defined in the *time.sim* file. It is recommended to use the same period as used for the model calibration (but not more than 10-15 years to avoid exhausting model runtimes). Also make sure that the number of years to skip (parameter *nyskip* in the file *print.prt*) is set correctly for printing the simulation output (typically *nyskip* = 3).
2. That the model is sufficiently parameterised (e.g. *landuse.lum* parameters, initial soil nutrient values, correct operation schedules in the *management.sch* file corresponding to the defined simulation period).
3. That the model is sufficiently calibrated (i.e. a *calibration.cal* file with calibrated parameter values must be provided).
4. That the model prints only the relevant outputs required to calculate the optimisation objectives (irrelevant model outputs would consume unnecessary storage capacity and run time). If an optimisation objective requires channel outputs in daily resolution (e.g. low or high flow indicators), you must define the channel of interest (usually the outlet channel) in the *object.prt* file.
5. That you have removed unnecessary files and folders (SWAT+ output files can be extremely large).
6. That the SWAT+ master file (*file.cio*) lists all required input files, including those mentioned above (*time.sim*, *cal_parms.cal*, *calibration.cal*, *object.prt*).
7. That one SWATmeasR project file (<project_name>.measr) is included. The SWATmeasR project must contain appropriate settings for each of the NSWORMs considered. It is strongly recommended to check the plausibility of each individual NSWORM scenario prior to the optimisation.

The authors of this report thought several times that they had made all the settings in their own model, but then they were proven wrong. As a full optimisation run with SWAT+ models usually takes several weeks of time, double-checking is strongly recommended. A demo txt version from CS1 is available in the OPTAIN cloud (WPs & Tasks/WP5/SWAT+ model setups/CS1).

4.3.2. Adjust the *SWAT.R* script

The *SWAT.R* file is the script that is responsible for running SWAT+ within CoMOLA. When called, it implements the NSWORMs at individual sites within the study area according to the CoMOLA genome, runs the SWAT+ model and calculates the optimisation objectives. The optimisation targets are calculated based on a set of predefined indicator functions (stored in the *calc_opt_indis.R* file). Ensure that the correct indicator functions are used to calculate the objectives relevant to your case study. Table 4.1 lists all available indicators and their functions at the time of writing this report.

Table 4.1: Indicator functions available for the optimisation. Note that some functions require the variable *x* to be specified as an integer in square brackets; other functions require the indicator name (*ind*); and one function requires both. Functions using the HRU output files as input (e.g. `ind_hru_wb_aa()`) have the option of specifying the area considered in the calculation. Parameter *area* can be either 'basin' (considering all HRUs in the basin) or 'agr' (considering only cropland HRUs). The parameter *period* can be specified in function `ind_hru_mon_wb()` to include only the months of interest (as integer values, e.g. `c(5:9)` for the months May to September) in the calculation.

function	<i>ind_cha_aa(path, channel) [x]</i>	
<i>x</i>	indicator	description [unit]
1	Q_mean	mean discharge [m ³ /s]
2	Nload	total N load [kg/yr]
3	Pload	total P load [kg/yr]
4	Sedload	total sediment load [tons/yr]
function	<i>ind_cha_day(path, channel, ind) [x]</i>	
<i>x</i>	indicator (<i>ind</i>)	description [unit]
1	<i>Q_max</i>	maximum daily discharge [m ³ /s]
2	<i>Q_max_aa</i>	average maximum daily discharge of each year [m ³ /s]
3	<i>Q_p95</i>	95 percentile daily discharge [m ³ /s]
4	<i>Q_p90</i>	90 percentile daily discharge [m ³ /s]
5	<i>Q_p50</i>	50 percentile daily discharge [m ³ /s]
6	<i>Q_p10</i>	10 percentile daily discharge [m ³ /s]
7	<i>Q_p05</i>	5 percentile daily discharge [m ³ /s]
8	<i>Q_min</i>	minimum daily discharge [m ³ /s]
9	<i>Q_min_aa</i>	average minimum daily discharge of each year [m ³ /s]
10	<i>Q_maxmin</i>	Q_max/Q_min ratio []
11	<i>Q_maxmin_aa</i>	Q_max_aa/Q_min_aa ratio []
12	<i>Q_low_days</i>	frequency daily discharge is below low flow threshold []
13	<i>Q_high_days</i>	frequency daily discharge is below high flow threshold []
14	<i>Nconc_days</i>	frequency total N concentrations is below threshold []
15	<i>Pconc_days</i>	frequency total P concentrations is below threshold []
16	<i>Sedconc_days</i>	frequency total sediment concentrations is below threshold []
function	<i>ind_hru_aa_nb(path, area) [x]</i>	
<i>x</i>	indicator	description [unit]
1	N_loss	average annual N loss from land objects [kg N/ha,yr]

2	P_loss	average annual P loss from land objects [kg P/ha,yr]
3	Sed_loss	average annual sediment loss from land objects [tons/ha,yr]
4	N_loss_ratio	average annual N loss/input ratio []
5	P_loss_ratio	average annual P loss/input ratio []
function	<i>ind_hru_aa_wb(path, area) [x]</i>	
x	indicator	description [unit]
1	sw	average annual total soil moisture [mm]
2	sw300	average annual soil moisture in top 30 cm [mm]
3	perc	average annual percolation [mm]
function	<i>ind_hru_mon_wb(path, ind, period, area)</i>	
	indicator (<i>ind</i>)	description [unit]
	<i>sw</i>	average soil moisture of the whole soil profile in period of interest [mm]
	<i>sw300</i>	average soil moisture (top 30cm) in period of interest [mm]
function	<i>ind_bsn_aa_crp(path, grain_units, ind, crops_sel)</i>	
	indicator (<i>ind</i>)	description [unit]
	<i>grain_units</i>	average annual sum of grain units in whole basin []
	<i>cropland</i>	average annual area of cropland in whole basin [ha]

Below are some examples of how to use these functions in the *SWAT.R* file. CoMOLA aims to maximise fitness values for all objectives. Therefore, if an objective is to be minimised (such as the average nutrient load or the frequency of high nutrient concentrations at the catchment outlet), the corresponding indicator function must be multiplied by -1. Please note that the parameter *path* must always be set to *txt_path*.

Example 1:

Optimisation objective 1 = Pload

- choose function `ind_cha_aa()`
- in file *print.prt*, indicate to print the average annual *channel_sd* output file
- define your channel of interest
- specify variable x in squared brackets (here x = 3)
- multiply with -1 as loads should be minimised

```
fit1 <- ind_cha_aa(path = txt_path,
                 channel = 'cha0926')[3] * -1
```

Example 2:

Optimisation objective 2 = frequency of days with streamflow > lowflow threshold

- choose function `ind_cha_day()`
- define your channel of interest
- in file *object.prt*, indicate to print this channel's output to file *cha_day.out*
- define parameter *ind* (here: 'Q_low_days')
- define a meaningful lowflow threshold value (*threshold_lowQ*), such as the observed average minimum daily discharge of each year of the simulation period)
- specify variable *x* in squared brackets (here *x* = 12)

```
fit2 <- ind_cha_day(path = txt_path,
                   channel = 'cha0926',
                   ind = 'Q_low_days',
                   threshold_lowQ = 0.0344)[12]
```

Example 3:

Optimisation objective 3 = soil water (top 30cm) for period May to June in cropland

- choose function `ind_hru_mon_wb()`
- in file *print.prt*, indicate to print the monthly *hru_wb* output file
- define indicator of interest (here 'sw300')
- define period of interest (provide the numbers for the months of the year)
- define area of interest (can be either 'basin' for whole basin area or 'agr' for cropland HRUs only)
- as the period consists of multiple months (May and June), calculate the mean value of this period using `mean(as.numeric())`

```
fit3 <- mean(as.numeric(ind_hru_mon_wb(path = txt_path,
                                       ind = 'sw300',
                                       period = c(5:6),
                                       area = 'agr')))
```

Example 4:

Optimisation objective 4 = grain unit sum for the whole basin

- choose function `ind_bsn_aa_crp()`
- define the crop types to be considered (here all crop types are used that are listed in R object *grain_units*)
- define the indicator of interest (here 'grain_units')
- define crop-specific grain units (here the R object *grain_units* is defined for the same-named function parameter); the R object *grain_units* with your crops of interest must be specified before defining the optimisation objective

```
grain_units <- data.frame('wbar' = 1.163,
                          'csil' = 1.071,
                          'wwht' = 1.209,
                          'wira' = 1.429,
                          'barl' = 1.163,
                          'akgs' = 0.682,
                          'wiry' = 1.174,
                          'sgbt' = 1)

fit4 <- ind_bsn_aa_crp(path = txt_path,
                      crop_sel = names(grain_units),
                      ind = 'grain_units',
                      grain_units = grain_units)
```

4.3.3. Configure the master file (*config.ini*)

The *config.ini* file lists all parameters which are relevant to run an optimisation with CoMOLA. To set up the CoMOLA runs, the parameter sections [config_model] and [config_optimization_algorithm] have to be adjusted. In the section [config_model] the paths to the R and python executable files must be set (*file_path_R* and *file_path_Python*, respectively). An easy way to find the correct path to the R executable file (which is also used by RStudio) is to open RStudio and execute the command `R.home('bin')` in the console. This returns the path to the R executable, which must be assigned to the parameter *file_path_R* in the *config.ini*.

```
> R.home('bin')
#> [1] "C:/Users/schuerz/AppData/Local/Programs/R/R-4.2.2/bin/x64"
```

The identification of the path to the correct python executable file was shown in the *miniconda* setup in section 4.1.2. The execution of the command `where python` in the *Anaconda* prompt (see Figure 4.6) returns the path to the python version 3.11.9 which was installed before. The path can be copied and assigned to the parameter *file_path_Python* in the *config.ini*.

Of the optimisation parameters, only two need to be adjusted, *pop_size* and *max_generations*.

pop_size defines the size of the population (i.e. the number (n) of individual NSWRM plans within a population). The population size should be large enough to ensure sufficient diversity between individuals, to accurately approximate the true Pareto front, and to prevent premature convergence to local optima by providing a broader genetic pool and more opportunities for crossover and mutation. Common practice suggests population sizes in the range of 50 to 500 for many optimisation problems (e.g. Coello Coello et al., 2007; Deb et al., 2002; Zitzler et al., 2001), but this can vary widely. Defining the population size requires consideration of problem complexity, computational resources, and empirical testing to find a balance that

ensures efficient and effective optimisation. As the SWAT+ models in OPTAIN require long computation times to evaluate the fitness of an individual NSWRM plan (~10 to 30 minutes for a 10-year simulation period), we do not have the time and computational resources for in-depth empirical testing in any of the case studies. Based on our experience from previous land use optimisation studies (e.g. Kamamia et al., 2022; Verhagen et al., 2018), we suggest setting *pop_size* to a value of 100. At the start of the optimisation, CoMOLA would then create 100 copies of the *models* folder in which the SWAT+ model can be run in parallel. Make sure that you have enough disc space in your CoMOLA environment. For the first test runs, *pop_size* should not be greater than 5.

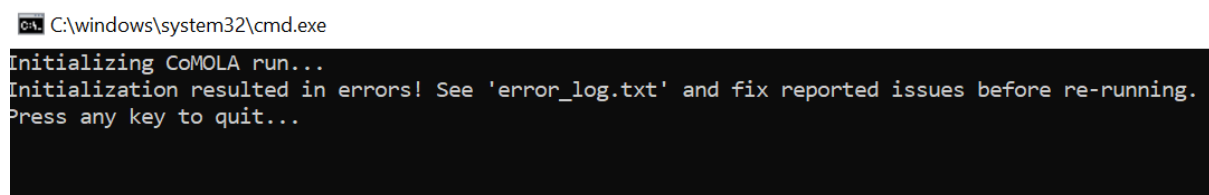
max_generations defines the maximum number of generations (i.e. the number of iterations in which a new offspring population of NSWRM plans is generated and evaluated during the optimisation). While *pop_size* must be large enough to ensure good coverage of the search space, *max_generations* must be large enough to allow the process of refining and improving upon the existing solutions to enhance their fitness. Due to the limited resources mentioned above, we suggest limiting the number of generations to 150. This would result in a CoMOLA routine with 15,100 SWAT+ model runs, which could take 7 to 30 days to compute for an average-sized SWAT+ model, depending on the resources available (in particular the type of processor and the number of cores to be used for parallel processing). For initial testing, it is recommended to set *max_generations* to 2.

In summary, there are four settings to be made in the *config.ini* file:

- (1) *file_path_R* = path to your R executable
- (2) *file_path_Python* = path to your python executable (version 3.11.9)
- (3) *pop_size* = 100 (for initial tests *pop_size* = 5)
- (4) *max_generations* = 150 (for initial tests *pop_size* = 2)

4.3.4. Starting the optimisation (*run_comola.bat*)

The CoMOLA routine starts with a double click on the batch file *run_comola.bat*. This opens the Windows command prompt and performs some internal checks on important optimisation requirements (e.g. if the correct version of the SWATmeasR R package has been used to build the SWATmeasR project). If any of the requirements are not met, an error message is printed to the file *error_log.txt* and the routine is aborted (Figure 4.9). All problems must be solved before the actual CoMOLA routine is started.



```

C:\windows\system32\cmd.exe
Initializing CoMOLA run...
Initialization resulted in errors! See 'error_log.txt' and fix reported issues before re-running.
Press any key to quit...
  
```

Figure 4.9: Prompt indicating errors at the start of CoMOLA.

If there are no errors, the command prompt will ask for the number of cores to be used to run CoMOLA (Figure 4.10). It is strongly recommended to use a High Performance Computing (HPC) cluster for Windows (the current SWAT+ executable used in OPTAIN has not been compiled for Linux). An HPC would allow CoMOLA (actually the SWAT+ model) to run on more than 20 nodes. The optimal number of cores is equal to `pop_size`. Then all individuals of a generation would run in parallel. If no HPC is available, a single machine can be used. However, this machine should be powerful and not be used for other tasks while the optimisation is running.

```
C:\windows\system32\cmd.exe
Initializing CoMOLA run...
Start of CoMOLA routine
-----
Define the number of cores for running CoMOLA:
```

Figure 4.10: Prompt indicating a successful start of the CoMOLA routine.

Running SWAT+ models can take a while (be patient with a log text such as shown in Figure 4.11). Errors will be printed in the command prompt and in the `*_optimisation_log.txt` file. Post the error message on the UFZ GitLab or GitHub if you cannot fix the problem yourself. A successful run of CoMOLA will automatically close the command prompt.

```
C:\windows\system32\cmd.exe
30-05-2024 17:55:26 | Run external models ...
30-05-2024 17:55:26 | Start model C:\+PAPER_WORK+\Opti-Tool\CoMOLA_CS1_240503\models\SWAT\SWAT.R
30-05-2024 17:55:26 | Start model C:\+PAPER_WORK+\Opti-Tool\CoMOLA_CS1_240503\models_3\SWAT\SWAT.R
30-05-2024 17:55:26 | Start model C:\+PAPER_WORK+\Opti-Tool\CoMOLA_CS1_240503\models_2\SWAT\SWAT.R
30-05-2024 17:55:26 | Start model C:\+PAPER_WORK+\Opti-Tool\CoMOLA_CS1_240503\models_1\SWAT\SWAT.R
30-05-2024 17:55:26 | Start model C:\+PAPER_WORK+\Opti-Tool\CoMOLA_CS1_240503\models_4\SWAT\SWAT.R
```

Figure 4.11: Prompt indicating SWAT+ models running in parallel.

4.4. Analysis of results

For each successful optimisation run, CoMOLA creates five output files in the `output` folder, of which only two files are relevant for the analysis of results: `*_optimisation_log.txt` (also called 'log file', * indicating the date and time of the start of the optimisation run) and `*_individuals_file.csv` (also called 'ind file').

The log file contains all the information printed to the command prompt during the optimisation and can be used to check for any errors that may have occurred. To do this, simply open the file in a text editor and search for the term 'Error'. At the end of this file, the user can find the final set of Pareto optimal solutions (fitness

values and genomes). The genomes and fitness values of all NSWORM plans tested in the optimisation can be found in the ind file.

As the output format in these files may not be convenient for users, it is recommended to access the optimisation results using the R script *CoMOLA_postprocessing.R* provided in the folder *output_analysis*. Running this file will extract and save the genomes and the fitness values of the Pareto optimal NSWORM plans in the files *pareto_genomes.txt* and *pareto_fitness.txt*, respectively. In addition, the hypervolume metric proposed by Zitzler and Thiele (1999) is computed to evaluate the different solution sets for each generation. Hypervolume is a widely accepted multi-objective performance metric that measures both convergence and diversity on a single scale without requiring knowledge of the true Pareto front for comparison (Jiang et al., 2014). It represents the volume in the objective space dominated by the set of solutions for a given reference point (here we use the origin of the coordinates). Thus, higher values of the hypervolume indicate that the solutions are closer to the true Pareto front and, at the same time, that they are more evenly distributed in the objective space (Jiang et al., 2014). The file *CoMOLA_postprocessing.R* can be used to plot the evolution of hypervolumes over generations. A converging curve (see an example in Figure 4.12) indicates sufficient exploration of the solution space. If the curve does not converge, this indicates that it is worth repeating the optimisation with a higher number of generations (parameter *max_generations* in the file *config.in*).

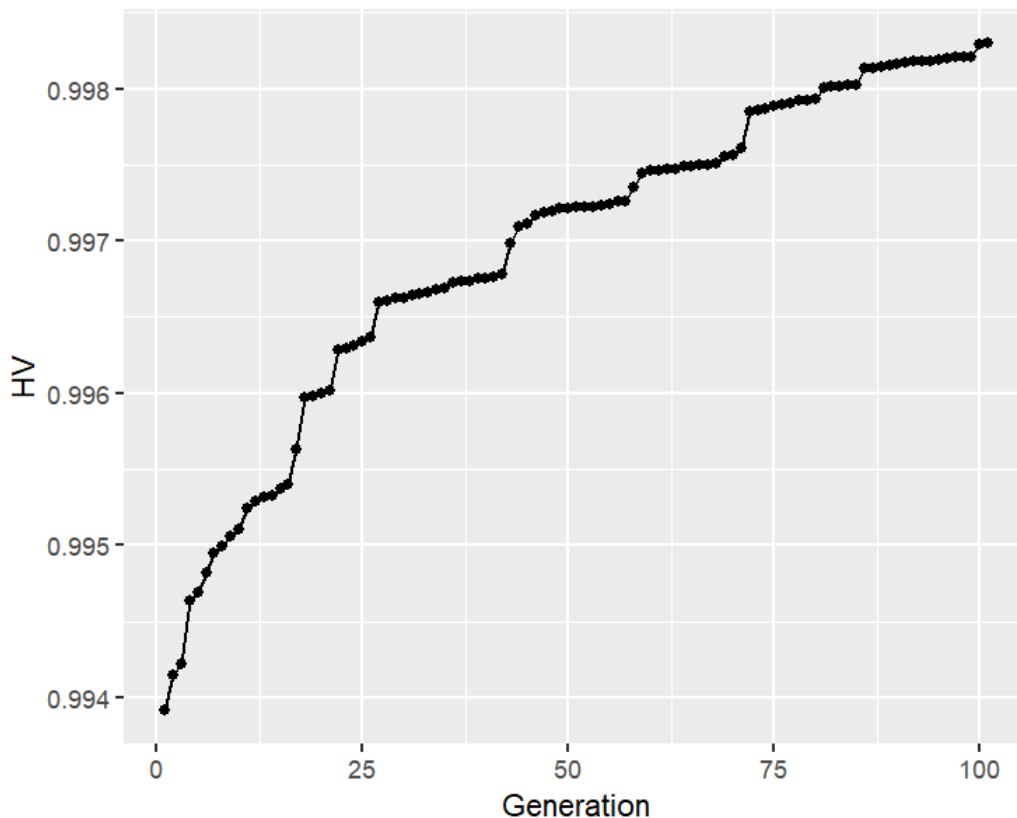


Figure 4.12: Evolution of hypervolumes over generations. Example plot using the *CoMOLA_postprocessing.R* file.

Eventually, the user can create a simple Pareto plot for the set of best solutions (see for example Figure 4.13). To do this, meaningful names should be defined for the different objectives at the beginning of the R script (corresponding to the indicator functions used for fit1, fit2, etc. in file *SWAT.R*, section 4.3).

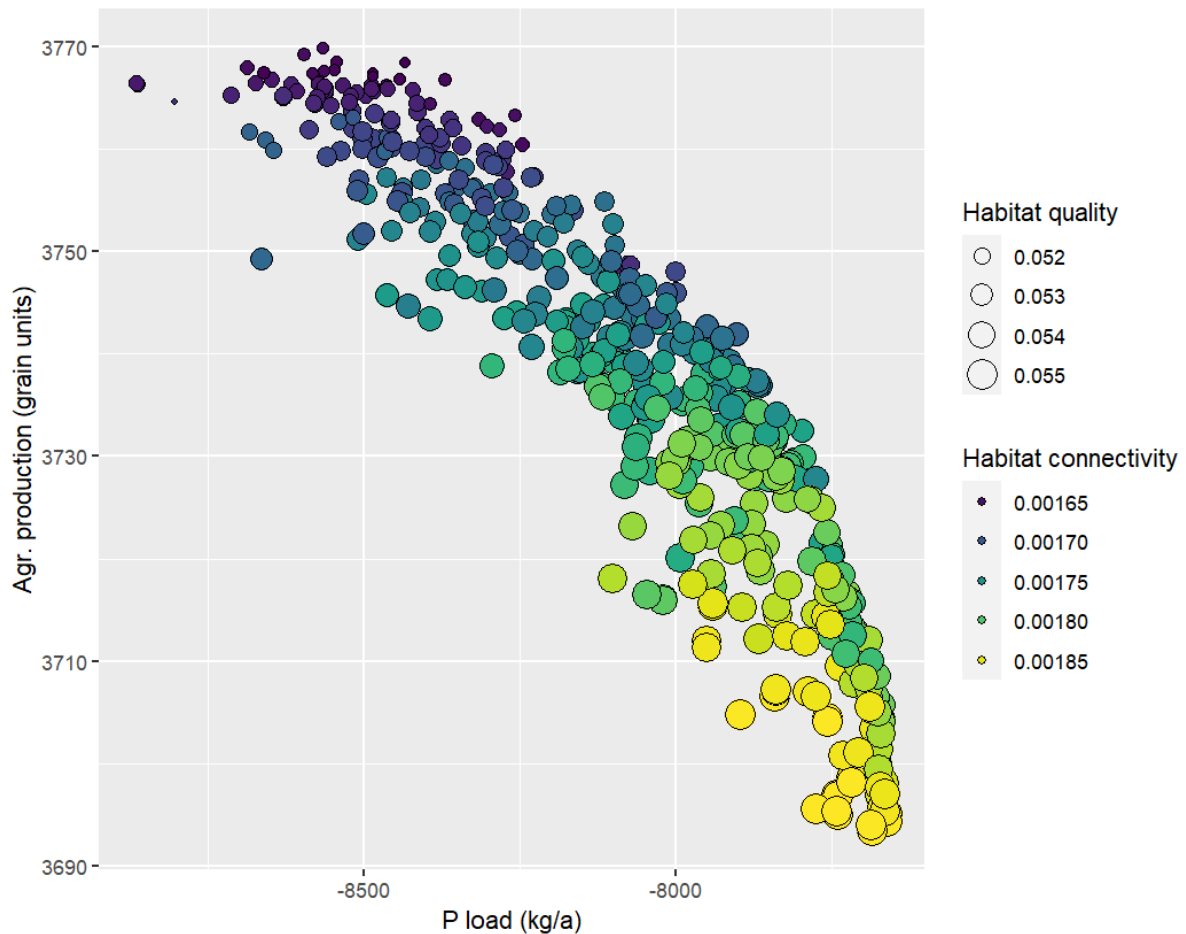


Figure 4.13: Scatterplot of Pareto optimal solutions created with the *CoMOLA_postprocessing.R* file (example plot).

The R code to post-process (e.g. visualise, filter) Pareto optimal solutions will be extended in the next phase of the project and published in Deliverable D5.2 ‘Report & Code for post-processing of optimisation results’.

5. References

- Chang, W., 2022. R6: Encapsulated Classes with Reference Semantics. <https://r6.r-lib.org>, <https://github.com/r-lib/R6/>.
- Coello Coello, C. A., Lamont, G. B., Van Veldhuizen, D. A., 2007. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer.
- Cord, A.F. et al., 2017. Towards systematic analyses of ecosystem service trade-offs and synergies: Main concepts, methods and the road ahead. *Ecosystem Services*, 28: 264-272. DOI:10.1016/j.ecoser.2017.07.012
- Deb, K. et al., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2): 182-197. DOI:10.1109/4235.996017
- Garrett, A., 2012. *inspyred* (Version 1.0.1) [software]. Inspired Intelligence. Retrieved from <https://github.com/aarongarrett/inspyred> [accessed 27.05.2014]
- Jiang, R. et al., 2021. A Hybrid Multi-Objective Optimization Method Based on NSGA-II Algorithm and Entropy Weighted TOPSIS for Lightweight Design of Dump Truck Carriage. *Machines*, 9(8): 156. DOI:10.3390/machines9080156
- Jiang, S. et al., 2014. Consistencies and contradictions of performance metrics in multiobjective optimization. *IEEE Trans. Cybern.* 44 (12), 2391-2404. DOI:10.1109/TCYB.2014.2307319
- Kaim, A., Cord, A.F., Volk, M., 2018. A review of multi-criteria optimization techniques for agricultural land use allocation. *Environmental Modelling & Software*, 105: 79-93. DOI:10.1016/j.envsoft.2018.03.031
- Kaim, A., Strauch, M., Volk, M., 2020. Using Stakeholder Preferences to Identify Optimal Land Use Configurations. *Frontiers in Water*, 2. DOI:10.3389/frwa.2020.579087
- Krzeminska, D. & Monaco, F., 2022. Tailored environmental and socio-economic performance indicators for selected measures. Deliverable D2.2 of the EU Horizon 2020 project OPTAIN. DOI:10.5281/zenodo.7050653
- Marval, Š., et al., 2022. SWAT+ and SWAP retention measure implementation handbook. Deliverable D2.3 of the EU Horizon 2020 project OPTAIN. DOI:10.5281/zenodo.11232719
- Memmah, M.-M. et al., 2015. Metaheuristics for agricultural land use optimization. A review. *Agronomy for Sustainable Development*, 35(3): 975-998. DOI:10.1007/s13593-015-0303-4
- Piniewski, M. et al., 2024: Assessment of NSWORM effectiveness under current and future climate at the catchment scale. Deliverable D4.4 of the EU Horizon 2020 project OPTAIN. DOI:10.5281/zenodo.11233621
- Schürz, C., 2024a: SWATbuildR: An object connectivity based SWAT+ model builder, R package version 0.1.17. URL: <https://git.ufz.de/schuerz/swatbuildr>

- Schürz, C., 2024b SWATfarmR: Simple rule based scheduling of management operations in SWAT, R package version 4.0.3. URL: <https://github.com/chrisschuerz/SWATfarmR>
- Schürz, C. et al., 2022. SWAT+ modeling protocol for the assessment of water and nutrient retention measures in small agricultural catchments. Deliverable D4.2 of the EU Horizon 2020 project OPTAIN. DOI:10.5281/zenodo.7463395
- Strauch, M. et al., 2019. Constraints in multi-objective optimization of land use allocation – Repair or penalize? *Environmental Modelling & Software*, 118: 241-251. DOI:10.1016/j.envsoft.2019.05.003
- Zitzler, E., Laumanns, M., & Thiele, L., 2001. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. TIK-Report, 103.
- Zitzler, E. & Thiele, L., 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Trans. Evol. Comput.* 3 (4), 257–271. DOI:10.1109/4235.797969