Software Engineering

WS 2024/25, Assignment 04

Prof. Dr. Sven Apel
Lukas Abelt
Sebastian Böhm

Handout: 13.01.2025
Handin: 27.01.2025 23:59 CET

Organizational Section:

- The assignment must be accomplished by yourself. You must not collaborate with anyone. **Plagiarism leads to failing the assignment!**
- The use of generative AI tools for solving the assignment is not allowed and will be considered as plagiarism.
- The deadline for the submission is fixed. **A late submission leads to a desk reject of the assignment!**
- The submission must consist of a ".py" file fulfilling the following criteria:
  - You **used the provided python template** in the materials section on the course's CMS page.
  - Your name and matriculation number are included as specified by the template.
  - Do **not change the signatures of the methods provided**. If you change the signatures this will lead to **0 Points for the respective sub-tasks.**
  - Do **not use any further libraries or imports.**
  - Make sure the given test cases run on your implementation.
- Any violation of these submission format rules leads to a desk reject of the assignment.
- Make sure to read through the **General Information** section after the task descriptions!
- Questions regarding the assignment can be asked in the forum. Please do not share any parts that are specific to your solution, as we will have to count that as attempted plagiarism.
- If you encounter any technical issues, inform us immediately.

Task 1 [10 Points]

Implement an algorithm that creates a CART from performance data as presented in the lecture (Chapter 6, Slides 44-50). The CART must be implemented by yourself, you must not take a prefabricated Python implementation.
The format of the sample data, the representation of the CART and the termination criteria are described below in the ”General Information” section.
If two split options are equally good, we use the alphabetic ordering of feature names as tie breaker.

Task 2 [5 Points]

Implement an algorithm that creates a Performance Influence Model (PIM), including feature interactions, as presented in the lecture (Chapter 6, Slides 53-61).
The format of the sample data, as well as additional information how to build the performance model are described below in the ”General Information” section.

Task 3 [10 Points]

In the next task you will work with CARTs and Performance Influence Models. More information in which format these are provided are described below in the ”General Information” section.

a) Given a CART and a (potentially partial) configuration, determine the predicted performance according to the CART. [2.5 Points]

b) Given a CART and a sample set, calculate the error rate. The error rate is computed by predicting the value with the given CART for every sample in the sample set and then averaging the differences between the predicted and original performance. [2.5 Points]

c) Given a Performance Influence Model (PIM), a partial configuration and a feature model, output the optimal **full** configuration and its' predicted performance. For this task assume that the PIM models the *cost* of a specific configuration. Therefore, the optimal configuration is the one with the lowest cost. The configuration you return must be in line with the partial configuration.
The format of feature models is described below in the ”General Information” section.
Note: For the scope of this assignment you can assume that the feature models will be small enough to efficiently generate all possible valid configurations. [5 Points]

General Information

In this assignment, we analyze a non-functional property, performance, of a fictional configurable software system.

We provide you with a few test cases for one system. **However, your solution must also work with performance data from other software systems that have other feature names!**

Grading

Your submission will be graded based on the following criteria:

- Your submission must be a single *.py* file that has the same functions as the provided template file (May result in 0 points if violated)
- No additional imports are allowed, **otherwise your submission is invalid and leads to a desk reject.**
- No additional libraries are allowed, **otherwise your submission is invalid and leads to a desk reject.**
- No changing of signatures is allowed, **otherwise your submission is invalid and leads to a desk reject.**
- We run tests (provided ones + **additional tests**) against your submission. Points are awarded for every passed test.

Project Skeleton

You have to implement your solution using the provided project skeleton "Template Assignment 04.zip" that can be downloaded from the CMS. Once unpacked, the project skeleton has the following structure:

```
│
├── Assignment04.py
├── Assignment04_Example_Tests.py
├── Performance_01.csv
├── Performance_02.csv
└── Performance_03b.csv
```

In order to implement the project and run the tests you need to install the following additional packages using `pip`:

- `pandas`[1]
- `findimports`[2] (Required to run example tests)

The files have the following purpose:

- `Assignment04.py` – The main implementation file.
  - *This is the file where you implement your solution and subsequently upload to the CMS.*
  - This python file contains comments where your implementation goes, marked with "TODO"
- `Assignment04_Example_Tests.py` – A file that runs some basic tests that ensures that your implementation works.
  - Execute with `python Assignment04_Example_Tests.py` from the source folder of the project skeleton.
  - This python file executes some basic checks and one simple test case for each (sub-)task.
  - Make sure that your code passes all checks!
  - *The full evaluation we use for grading will include additional test cases!*
- `Performance_01.csv` – Example data for the test case of task 1.
- `Performance_02.csv` – Example data for the test case of task 2.
- `Performance_03b.csv` – Example data for the test case of task 3.

Running the provided test

As a running example, consider the tool "SECompress", a configurable command-line tool for compressing data. In addition, the data can be encrypted, signed, segmented, and/or time-stamped. All this functionality is modeled by the feature diagram in Figure 1.

There is a simple test-suite that runs a few sanity checks and one simple, basic test for each (sub-)task of the assignment. To execute it run `python Assignment04_Example_Tests.py` from the source folder of the project skeleton.

[1] `https://pypi.org/project/pandas/`
[2] `https://pypi.org/project/findimports/`

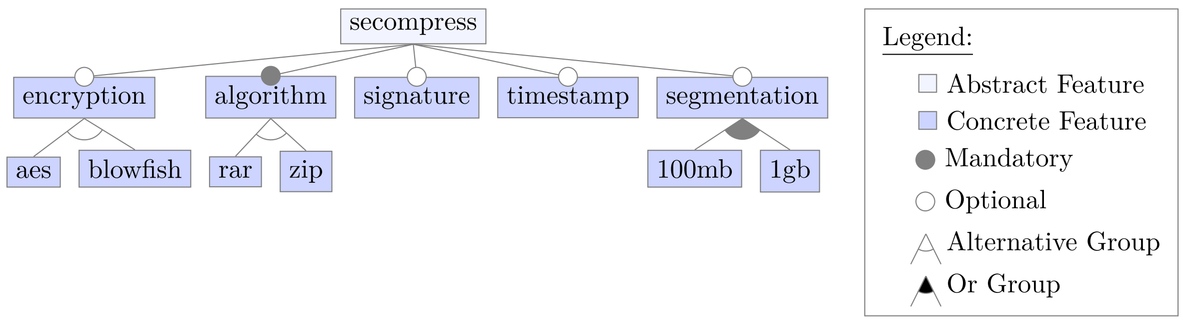

Figure 1: The Feature Diagram of "SECompress"

Important: To run the test file you need to install the `findimports` python package.

A valid submission should produce the following output (Assuming your name is "John Doe" and your matriculation number is "1234567"):

```
Your name: John Doe
Your matriculation number: 1234567
Import check: Passed
Task 1: passed
Task 2: passed
Task 3a: passed
Task 3b: passed
Task 3c: passed
```

Performance Data

Performance data is given as a CSV file and consists of entries for different configurations of the software system under analysis in a similar format as presented in the lecture (Chapter 6, Slide 44). The following shows an example for the "SECompress" tool:

```
Id,secompress,encryption,aes,blowfish,algorithm,rar,zip,signature,timestamp,
    ↪ segmentation,100mb,1gb,performance
0,1,0,0,0,1,1,0,0,0,0,0,0,750
1,1,0,0,0,1,1,0,0,0,1,1,0,773
2,1,0,0,0,1,1,0,0,0,1,0,1,770
3,1,0,0,0,1,1,0,0,1,0,0,0,750
4,1,0,0,0,1,1,0,0,1,1,1,0,773
```

In the performance data, two columns have a special meaning:

- `Id` – Numeric identifier for a specific configuration.
- `performance` – Measured performance data for a configuration

All other column names are the **feature names** of the system under evaluation. It is important that your solution uses the **exact same** feature names as provided in the CSV file at all times. Otherwise, your solution may be marked as incorrect.

Termination criterion for CART

When constructing a CART, we do not split a node further when at least one of the following conditions applies:

a) All possible splits are exhausted

b) All remaining splits result in either the XL or the XR node containing no configurations

c) All configurations of the current node have the same performance value

d) The *sum of squared error loss* of the parent node is strictly lower than 10.

Make sure that your CART construction adheres to these termination criteria.

CART Data Structure

For this assignment sheet we use the following internal data structure for a CART. We represent a CART as a python dict with exactly the entries as shown in the example below. This example CART has three nodes. The root node

”X”, and the two child nodes ”XL” and ”XR”. As you can see the child nodes only have a name and a mean but all other fields are set to None. A parent node also has a name and a mean but additionally a feature by which the split is performed, the error of the split and two successors.

```
cart = {
                "name": "X",
                "mean": 45.6,
                "split_by_feature": "aes",
                "error_of_split": 7.3,
                "successor_left":
                        {
                                "name": "XL",
                                "mean": 123.4,
                                "split_by_feature": None,
                                "error_of_split": None,
                                "successor_left": None,
                                "successor_right": None
                        },
                "successor_right":
                        {
                                "name": "XR",
                                "mean": 25.8,
                                "split_by_feature": None,
                                "error_of_split": None,
                                "successor_left": None,
                                "successor_right": None
                        }
}
```

Note: The `"error_of_split"` attribute of the cart refers to the *sum of squared error loss* as presented in the lecture.

Important: In our CART data structure, we always assume that `successor_left` considers that the `split_feature` of the parent was selected, while in `successor_right` it is deselected.

Performance Influence Model (PIM) Structure

For this assignment sheet, we use the following internal data structure to represent a PIM. Please ensure that your return types match the described structure.
PIMs are represented as python dictionaries. The keys of this dictionary are strings, which represent a single or multiple features. In case of an entry for multiple features, the individual features are sorted lexicographically and separated by the asterisk symbol(`*`).
The value for a corresponding string describes the influence of this feature or feature interaction. The base performance is represented by the value assigned to the empty string. The dictionary shown below represents the following PIM:

$$\Pi(A, B, C) = 100 + 20 \cdot A + 30 \cdot B - 15 \cdot AB$$

```
pim = {
        "": 100,
        "A": 20,
        "B": 30,
        "A*B": -15
}
```

PIM Construction

When constructing your PIM, start by calculating the influence of singular features first. Then, successively calculate the influence of feature interactions of increasing size. For example: Before calculating the influence of any interaction of three features, you should first caluclate **all** interactions of two features. Your PIM should only contain terms for **feature interactions of size of at most 3**.
For Task 2, you can assume that the performance data will always contain the *base configuration*, for which all features are deselected, in the row where the `Id` is `0`. Your PIM should only consider influences for features and interactions that can be directly observed from the measurements. If an influence cannot be calculated from the measurements given you can assume it to be 0 for any further calculations.

Feature Model Structure

For this assignment we use the following internal data structure to represent a feature model:
We represent feature models as recursive python dictionaries. A feature model is passed to the corresponding functions as a dictionary object. A full example of the dictionary representation of the feature model for the ”SECompress” tool can be found in the file `Assignment04_Example_Tests.py`, starting at line 93.

Each dictionary represents a feature and can contain the following entries:

- `"name"` – Name of the current feature
- `"groupType"` – Specifies the group type for the child features. Is either `"And"`, `"Or"` or `"Xor"`. Entry will not be present for leaf features.
- `"featureType"` – If present, specifies whether the current feature is optional (Value: `"Opt"`) or mandatory (Value: `"Mand"`). This entry will only be present for features within an `"And"` group.
- `"children"` – A list of dictionaries representing the child features of the current feature. Entry will not be present for leaf features.

Note: Our feature models do not include any cross-tree constraints.

Function Signatures

For the respective subtasks you have to follow the function signatures as they are provided in the template file. The docstrings in the template file describe the individual arguments in more detail.
You **must not** change the arguments or return types of the provided functions. You are allowed to add additional helper functions.