Java framework for Evolutionary Computation and Decision-Making (JECDM)

Michał Tomczyk **Miłosz Kadziński**

Institute of Computing Science
Poznan University of Technology, Poland

michal.tomczyk@cs.put.poznan.pl
www.cs.put.poznan.pl/mtomczyk/

Research scope + about me

My research interests include:

- multi-objective optimization (solving real-world problems)
- evolutionary algorithms,
- multi-criteria decision aiding (preference learning)

I'm primarily focused on developing **advanced, interactive metaheuristics for solving real-world optimization problems**.

Information Sciences 549 (2021) 178–199

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Decomposition-based co-evolutionary algorithm for interactive multiple objective optimization

Michał K. Tomczyk [a,*], Miłosz Kadziński [a]

[a] Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60-965 Poznań, Poland

Applied Soft Computing Journal 152 (2024) 111258

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Evolutionary algorithms for solving single- and multiple-objective political redistricting problems: The case study of Poland

Michał K. Tomczyk [a], Miłosz Kadziński

Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60-965 Poznań, Poland

Information Sciences 616 (2022) 157–181

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Interactive co-evolutionary multiple objective optimization algorithms for finding consensus solutions for a group of Decision Makers

Michał K. Tomczyk [a,*], Miłosz Kadziński [a]

[a] Faculty of Computing and Telecommunications, Poznan University of Technology, Piotrowo 2, 60-965 Poznań, Poland

Omega 68 (2017) 168–184

Contents lists available at ScienceDirect

Omega

journal homepage: www.elsevier.com/locate/omega

Evaluation of multi-objective optimization approaches for solving green supply chain design problems ☆

Miłosz Kadziński [a,*], Tommi Tervonen [b], Michał K. Tomczyk [a], Rommert Dekker [c]

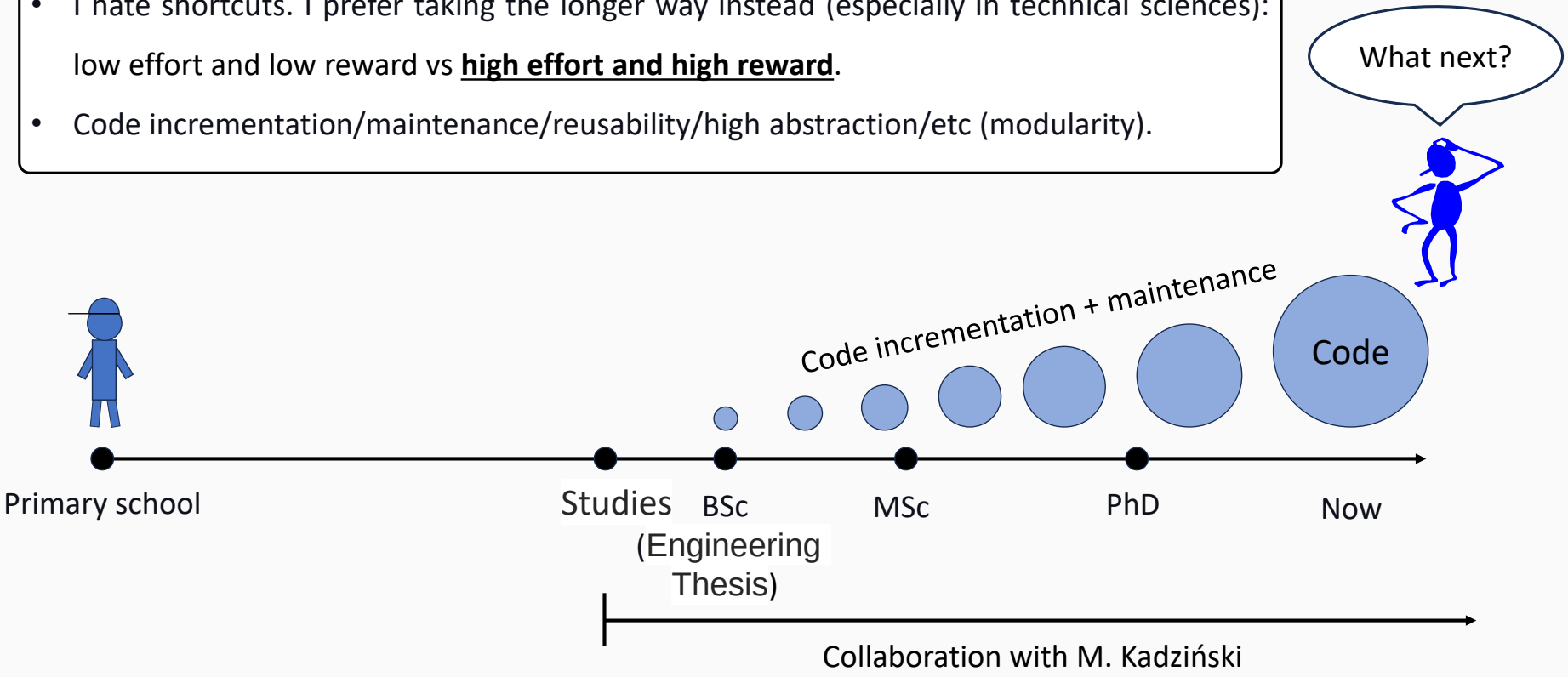
320 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 24, NO. 2, APRIL 2020

Decomposition-Based Interactive Evolutionary Algorithm for Multiple Objective Optimization

Michał K. Tomczyk and Miłosz Kadziński

Research scope + about me

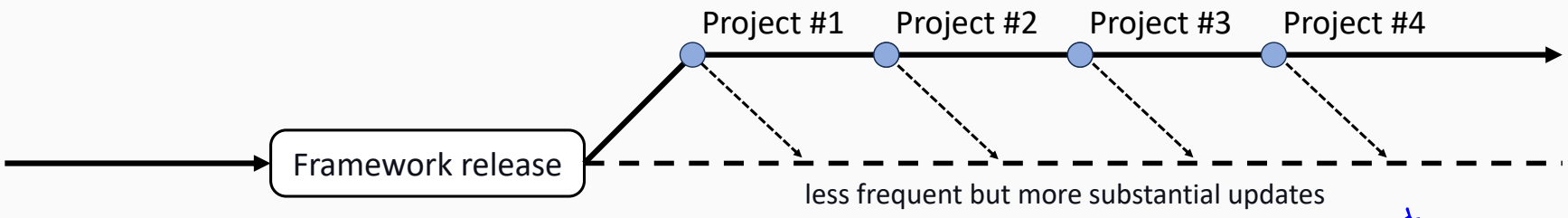
- Programming (data structures/algorithms/AI/etc.) has been one of my hobbies (?). interests since I was a kid. I notice many similarities between LEGO and programming ☺.
- I hate shortcuts. I prefer taking the longer way instead (especially in technical sciences): low effort and low reward vs **high effort and high reward**.
- Code incrementation/maintenance/reusability/high abstraction/etc (modularity).



Motivations

Motivation – a strategic decision:

- doing things simultaneously
- boost for own research
- creating own environment (sandbox) that will be shared (research transparency, reusability, self-promotion, potential collaboration)



- Next focus: other projects
- The future framework's updates will result from the progress made when realizing other projects (**less frequent but more substantial updates**).

About the framework

- The framework is written in Java (v. 1.0 uses Java 17). Motivations for Java are:
  - Need to maintain an extensive (> 1000 classes, > 150000 lines of code), but highly structured project (object-oriented programming, software engineering, etc.)
  - Fast (runtime, not startup)
  - Memory optimization or direct memory addressing is not of concern.
- High-level abstractions and code reusability...
- but still focused on efficiency in those places when the performance is critical.
- It follows good practices delineated by software engineering practitioners but can also be deemed high-quality from the viewpoint of specialists in algorithms and data structures.
- Self-sufficiency (low use of eternal libraries, currently only 4 small, technical libraries are used).
- The framework is maintained using the IntelliJ IDEA software (shared on GitHub and as copied & pasted sources).
- **Primary focus: architecture, not methods.**

About the framework

Instead of the product on the left, I am focused on developing a product on the right.

(+) Easy to comprehend

(-) Low reusability

(-) Low potential for extensions

(-) More difficult to comprehend

(+) High reusability

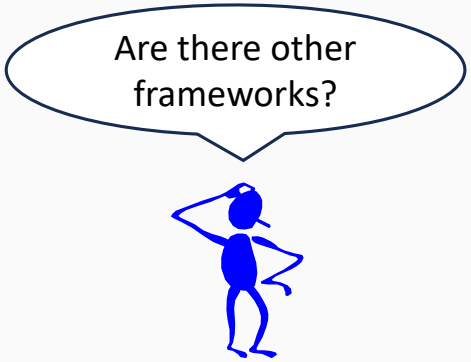
(+) High potential for extensions

The framework will be discussed and explained (how to use it) in a series of tutorials (PDF + accompanying source codes).

Thank you Denmark ☺

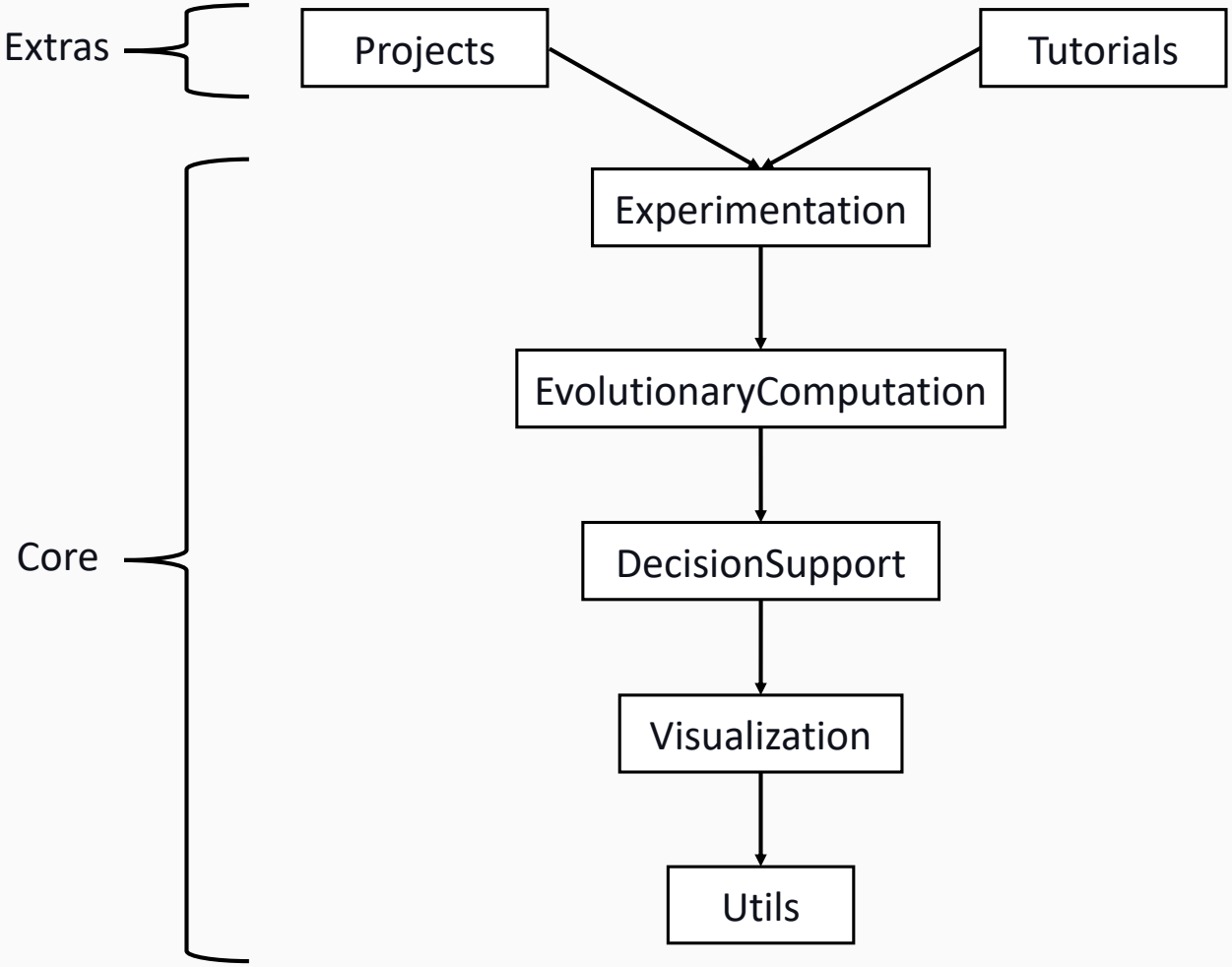
The pictures were borrowed from the LEGO store: https://www.lego.com/

About the framework



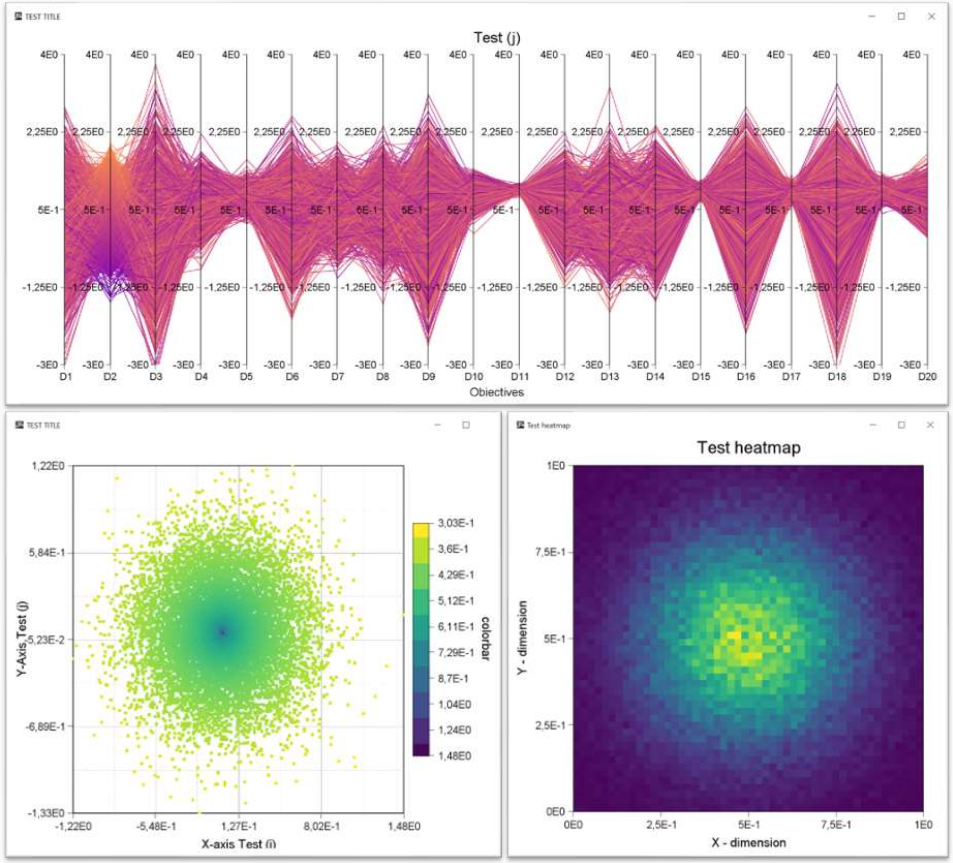
- For MCDA: the existing software is mainly focused on using existing methods (there are some exceptions, e.g., https://www.decision-deck.org/project/)
- EMO + MCDA: not at this scale (focus on integrating state-of-the-art from both fields)
- EMO: yes, but they have some flaws (strange architectural decisions, not so flexible for extensions, some hardcoding). For example:
  - Pymoo https://www.pymoo.org/news.html
  - MOEA Framework http://moeaframework.org/
  - JMetal https://jmetal.sourceforge.net/

Main modules

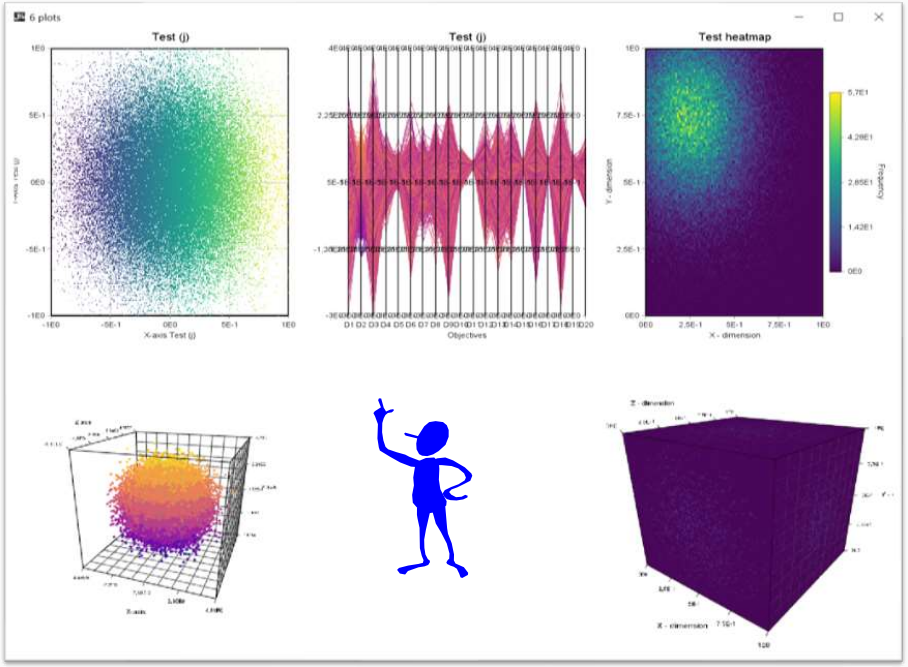


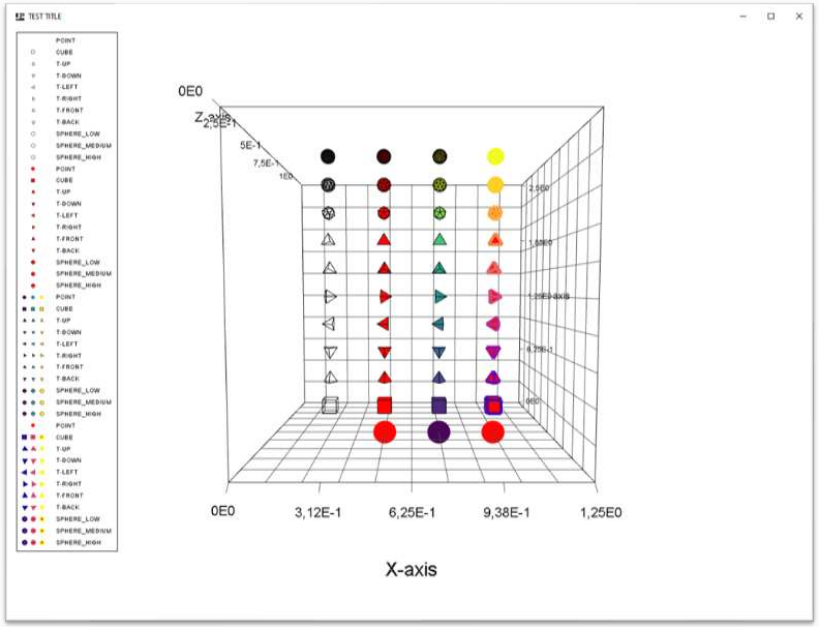
The *visualization* module

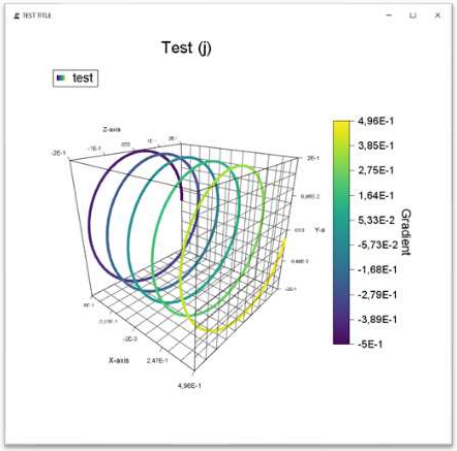
- Developed from scratch (almost).
- **High-level integration with other framework's components**.
- Supports 2D (Java Swing; processing on CPU) and 3D rendering (OpenGL; processing on GPU).
- Highly efficient. The main focus was to develop visualization components that can work well **with dynamically updating data**. The GUI processing is separated from more costly operations, which are executed on separate threads in a dedicated queuing system.
- **Highly flexible with a high potential for various adaptations**.

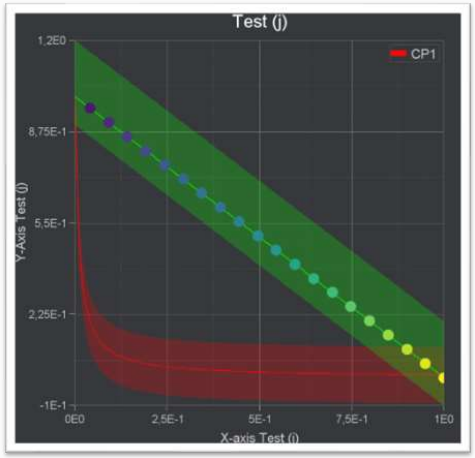


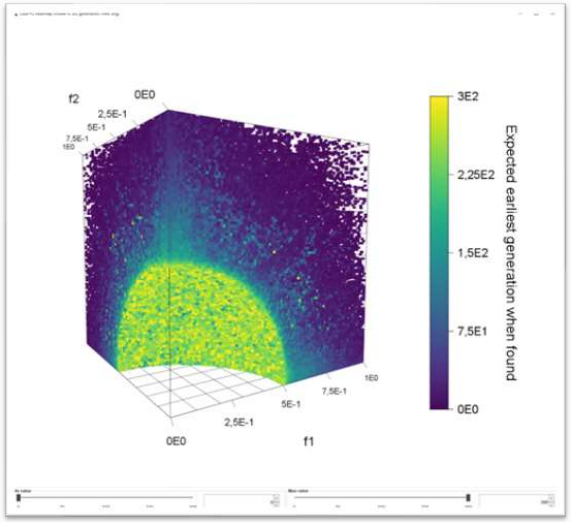
The *visualization* module





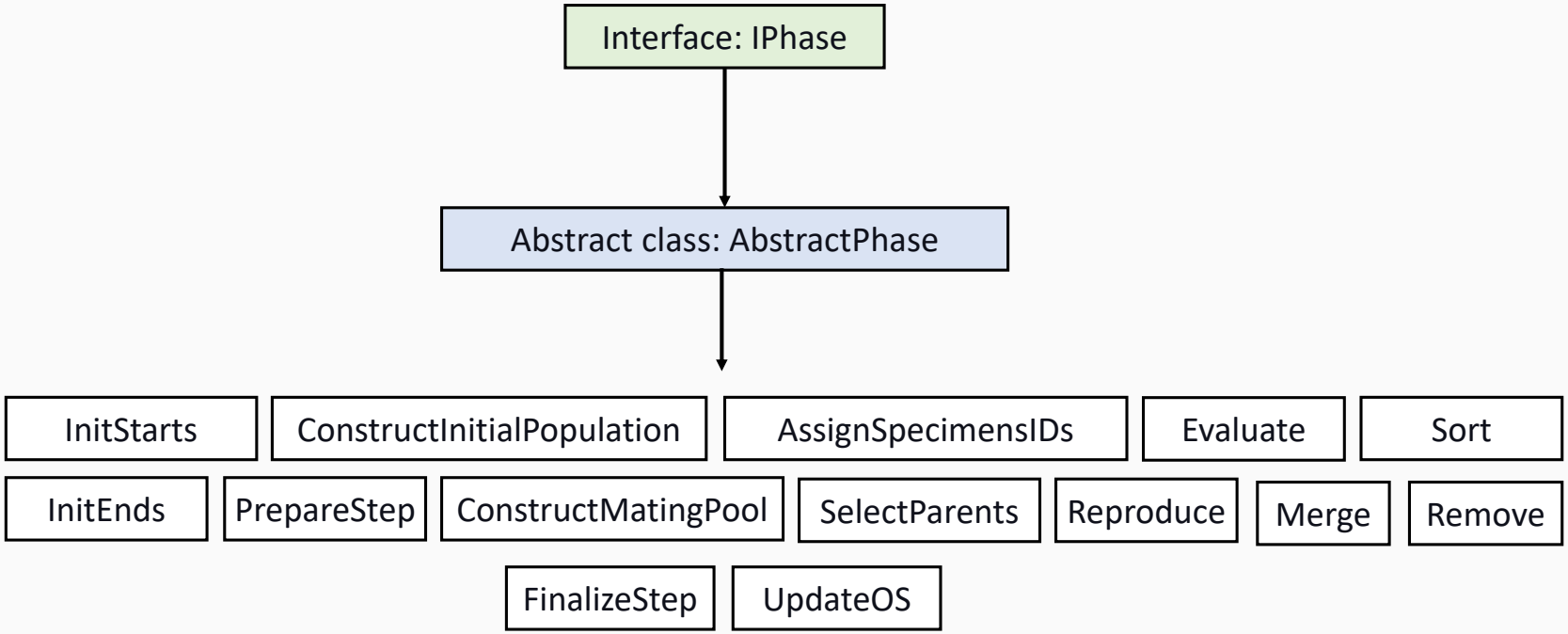






The *evolutionary computation* module

The evolutionary computation (algorithm) is built on the concept of phases processed by the upper-level class.



The *evolutionary computation* module

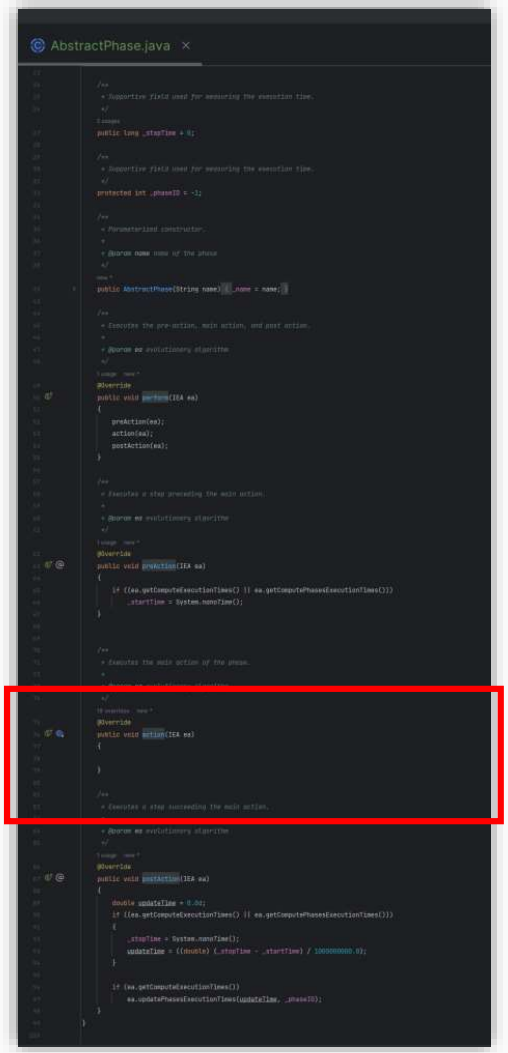
- The phases are managed and executed by the upper-level class (EA)
- The overall design should suit most cases well (regular implementations/algorithms)
- **The default implementations will probably do almost 90% of the job**. The programmer will usually have to overwrite some particular methods and decide upon the exclusion/inclusion of some phases.
- Some phases are somewhat technical, and there will be no reason to re-implement them (e.g., AssignSpecimensIDs); some must be implemented (e.g., solution evaluation).

```java
/**
 * Initializes the evolutionary process.
 */
new *
@Override
public void init()
{
    startMeasuringTime();
    _currentTimestamp = new EATimestamp( generation: 0,  steadyStateRepeat: 0);

    executePhase(PhasesIDs.PHASE_INIT_STARTS);
    executePhase(PhasesIDs.PHASE_CONSTRUCT_INITIAL_POPULATION);
    executePhase(PhasesIDs.PHASE_ASSIGN_SPECIMENS_IDS);
    executePhase(PhasesIDs.PHASE_EVALUATE);
    executePhase(PhasesIDs.PHASE_UPDATE_OS);
    executePhase(PhasesIDs.PHASE_SORT);
    executePhase(PhasesIDs.PHASE_INIT_ENDS);

    stopMeasuringTime();
}
```

```java
/**
 * Executes one step of the evolutionary algorithm.
 *
 * @param timestamp generation; steady-state repeat
 */
new *
@Override
public void step(EATimestamp timestamp)
{
    startMeasuringTime();
    _currentTimestamp = timestamp;

    executePhase(PhasesIDs.PHASE_PREPARE_STEP);
    executePhase(PhasesIDs.PHASE_CONSTRUCT_MATING_POOL);
    executePhase(PhasesIDs.PHASE_SELECT_PARENTS);
    executePhase(PhasesIDs.PHASE_REPRODUCE);
    executePhase(PhasesIDs.PHASE_ASSIGN_SPECIMENS_IDS);
    executePhase(PhasesIDs.PHASE_EVALUATE);
    executePhase(PhasesIDs.PHASE_MERGE);
    executePhase(PhasesIDs.PHASE_UPDATE_OS);
    executePhase(PhasesIDs.PHASE_SORT);
    executePhase(PhasesIDs.PHASE_REMOVE);
    executePhase(PhasesIDs.PHASE_FINALIZE_STEP);
    stopMeasuringTime();
}
```

The *evolutionary computation* module

Abstract class: AbstractPhase



Class: Evaluate

```java
/**
 * Phase main action.
 *
 * @param ea evolutionary algorithm
 */
new *
@Override
public void action(IEA ea)
{
    if (ea.getPopulation().isPopulationRequiringEvaluation())
        _evaluate.evaluateSpecimens(ea.getPopulation().getPopulation());
    if (ea.getPopulation().isOffspringRequiringEvaluation())
        _evaluate.evaluateSpecimens(ea.getPopulation().getOffspring());
}
```

Interface: IEvaluate

```java
/**
 * Interface for classes responsible for evaluating specimens.
 *
 * @author MTomczyk
 */
10 implementations   new *
public interface IEvaluate
{
    /**
     * Evaluates specimens.
     *
     * @param specimens array of specimens to be evaluated
     */
    2 usages   1 implementation   new *
    void evaluateSpecimens(ArrayList<Specimen> specimens);
}
```

The *decision support* module

- The decision support module is more like an addition to the evolutionary computation module than an equivalent component.
- The decision support module's strategic focus is to **facilitate the integration of both areas rather than attempting to implement all existing MCDA methods**. It will also focus mainly on **preference-learning** methods and value models.
- The module is implemented in the same spirit as the module for evolutionary Computation, and is oriented around **a top-level decision-support component that can handle various aspects of the decision-reaching process**, e.g., collecting preferences, requesting preferences, exploiting preference models, handling inconsistencies, evaluating solutions given the Decision Maker's preferences, and so on.

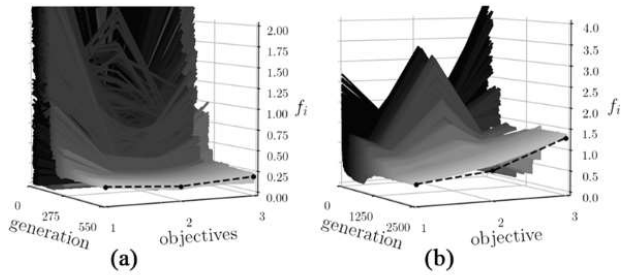


Fig. 5. Solutions constructed throughout evolutionary search by IEMO/D applied to DTLZ1 and WFG1 with $M = 3$ objectives and $w^{DM} = [0.3, 0.4, 0.4]$. (a) DTLZ1. (b) WFG1.

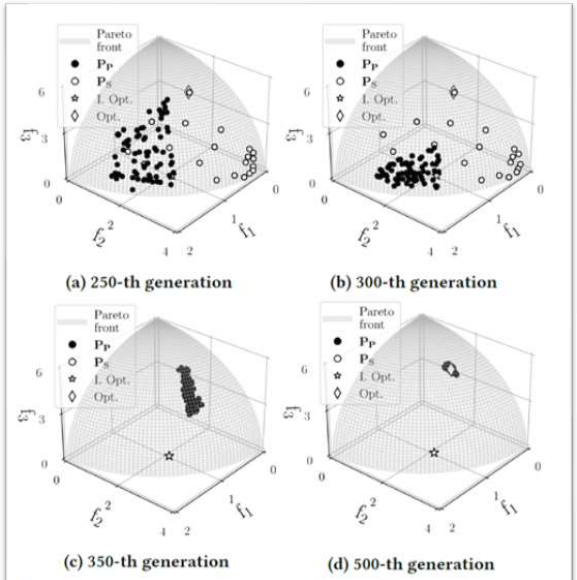
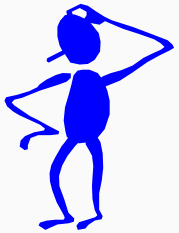


Figure 4: Populations constructed by IEMO/D_{CD} in different generations when applied to WFG4 with $M = 3$ and $DM = SC$.

The *decision support* module



Note that implementing a computational sandbox-like framework is not an easy task due to three reasons:

- Human dependency raises ambiguity when it comes to implementation.
- Many of the MCDA components are incompatible, which raises issues regarding generalization.
- Concepts<->methods: MCDA content is shifted more towards methods than concepts.

The *experimentation* module

- The module is implemented in the same spirit as the module for evolutionary computation.
- Template-like.
- Highly structured with default way of processing (3 levels, automation) and data organization provided.
- Efficiency, especially memory efficiency, is of high priority.

Defining 1-level read-only basic data

E.g., no. trial runs, data for optimization problems, or the main folder path

Executing 1-level operations

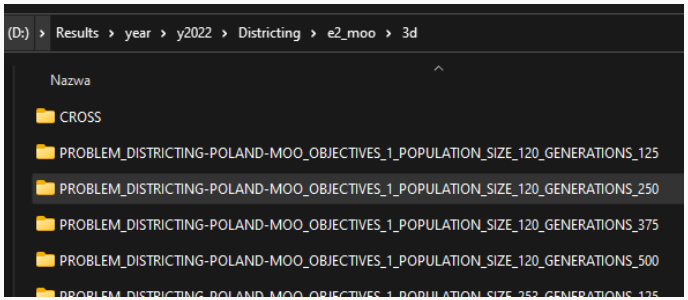
E.g., creating top-level folders

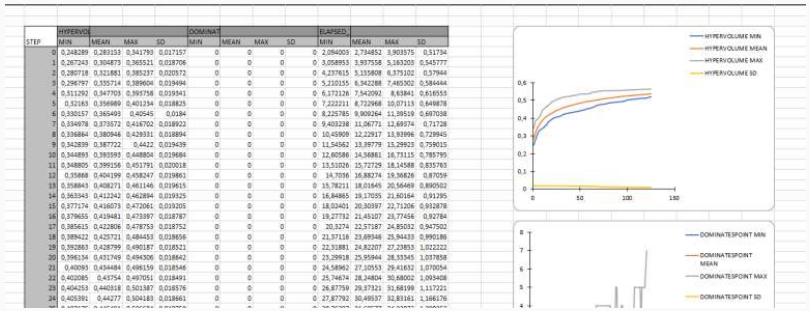
Defining experimental scenarios

A. **Problems:** DTLZ1-7, WFG1-9, .;
B. **Algorithms,** NSGA-III, NEMO-II, IEMO/D, ...
C. Objectives: 2-5;
D. Interactions: 5-10.
E. ...

The *experimentation* module

- Supports providing new performance indicators (implementing an interface)
- Supports parallelization of the 3-level processing.
- Supports providing new file processors (store results on disc)
- Supports providing new finalizers (construct experiment summary).





For each tuple that defines a scenario; (A, B, C, D, E, ...)

Defining 2-level (scenario-dependent) read-only data

E.g., performance indicators, the numer of generations, the population size, read-only operators, etc.

For each trial run (e.g., 100, 3 level):

1. **Instantiate** the algorithm
2. **Run the algorithm** and store the intermediate and final results

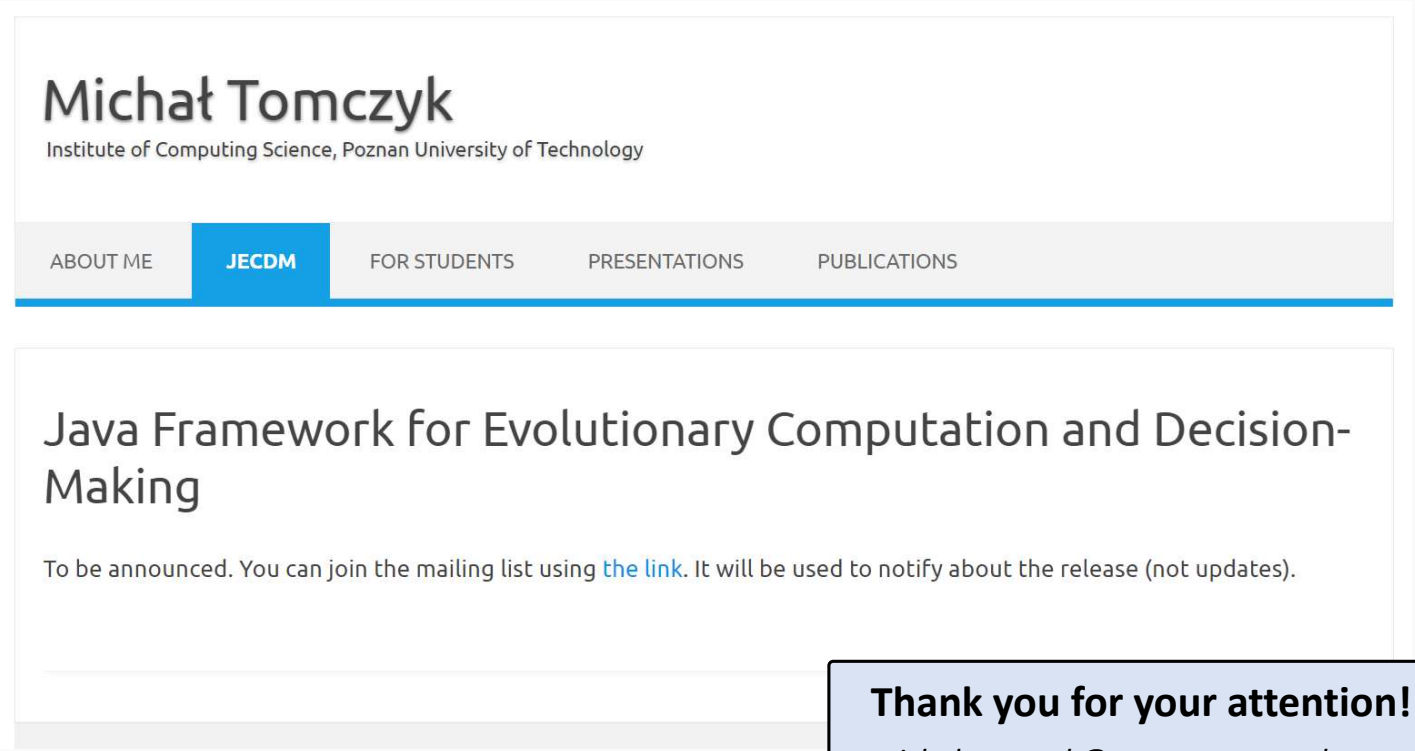
Finalize 3-level

Finalize 2-level

Finalize 1-level

Mailing list

The framework is currently under development, but I plan to release it in around 3 months. See my website: https://www.cs.put.poznan.pl/mtomczyk/index.php/jecdm, where you can join a mailing list that will be used to send the notification about the release.



Thank you for your attention! ☺

michal.tomczyk@cs.put.poznan.pl