# Project Description for Lecture Machine Learning for Software Engineering – SS 17

1. **Task Description**

   The goal of the project is to develop a running algorithm that finds (a) the optimal configuration for a single-objective optimization problem and (b) the Pareto front of optimal configurations for a multi-objective optimization problem. There are several subtasks you need to implement to achieve this goal:

   - Build a model (i.e., a representation of the optimization problem): The variables are given in the file [SYSTEM]_model.xml or [SYSTEM].dimacs. See the slides of the project introduction to know how to interpret and parse the files. A feature corresponds to a variable. A configuration is a complete assignment of the Boolean values True and False to each variable. That is, a configuration is a mapping from each variable to either true (if the corresponding feature is selected) or false. You might encode a configuration as a bit vector, in which each entry of the vector corresponds to a variable and has the domain: [0,1].
   - Build a method or use an existing tool to check whether a configuration is valid. The model files do not just list the variables of a system, but also possible constraints between them. This means that not all configurations are valid. There might be variables that cannot be simultaneously selected in a configuration. That is, if you have encode the model as a bit vector, there might be two entries in the vector that can never be set to 1 for the same configuration, because this would lead to an invalid configuration.
   - Build a fitness assessment function: You need a function to rate the quality (i.e., performance) of a configuration. To this end, you need to parse the files [SYSTEM]_feature.txt and [SYSTEM]_interactions.txt. In the multi-objective optimization case, there might be several of these files for the same system, each denoting another property to optimize. The feature files represent the performance values assigned to each feature/variable. That is, setting the corresponding variable to 1 or true, would increase the performance by the amount given in the file. Remember, the goal is to minimize the performance (lower is better). The interaction files represent the performance values assigned to combinations of features/variables. That is, when all features/variables are set to 1 or true, then the corresponding value must be added to the overall performance for the corresponding configuration. For example, if the line is: F2#F3: 23.5, then you need to add 23.5 to the fitness value of all configurations where the features/variables F2 and F3 are simultaneously set to 1 or true. If not all features/variables are set to 1 or true, then nothing needs to be added.
   - Build an algorithm to construct a solution (i.e., a configuration) and assess and rate its corresponding performance value. Here, you are entirely free to select an optimization approach that you have seen in the lecture or (better) construct one by yourself. You could use genetic algorithms, ant colony optimization, tabu search, or something else to do the optimization of configurations. In this algorithm, you will likely combine the methods you have implemented in the points above.
   - Some hints for your algorithm: You could either stay in the valid configuration space or work also with invalid configurations and make them valid later (i.e., fix them). This means that your tweak operation could always check for validity or you might find the configuration with minimal performance including invalid configurations and find the closest valid one. Even another idea is to define validity as another objective. That is, you could count the constraints that the current configuration breaks and try to minimize also this number.

Further useful information:

- For constraint and validity checking of a configuration, you should use an existing Python library. Search for SAT or CSP solvers or DIMACS format and you will find plenty of libraries that provide the functionality to check a Boolean formula (your model encoded as CNF) for validity. It is allowed and you are even encouraged to use this existing functionality in your code. Nevertheless, be aware that such validity checks are usually very time consuming. You might develop some heuristics for easy checks to filter easy cases or enable caching of invalid variable combinations.
- Submit either a Python script file or a Jupyter notebook file via email to norbert.siegmund@uni-weimar.de. Please make sure that you mention the participating team member with their full names and student id. .
- Deadline is Tuesday 15th of August 2017.