Module 4

Ryo Kimura

4 Detailed Models

In his module we will discuss methods to write more flexible optimization models which incorporate input data, modified solver parameters, and command line arguments.

4.1 DMAS Structure

The optimization model we implemented in module 3 was relatively simple and fairly inflexible. In contrast, most optimization models used in practice require some degree of flexibility; e.g., gracefully handling different input parameters, limiting the use of system resources, supporting certain solution queries, etc. A common way to achieve this is by separating the optimization model into four components, which we refer to as the *DMAS structure*:

- **D**ata: all "external" information that is needed to write down a specific MIP model
- **M**odel: a representation of the model that is to be solved
- **A**lgorithm: a procedure that transforms the state of the model from *Not Solved* to *Solved*, generating relevant information along the way
- **S**olution: a value for each decision variable that satisfies all of the constraints (and ideally has the "best" objective value)

MIP solvers (i.e., Gurobi and CPLEX) are written with this DMAS structure in mind; in particular, the API is designed so that each of these components can be modified and developed separately from the others.

4.2 Modifying the Data

Both Gurobi and CPLEX can read in MIP models that are encoded in LP and MPS formats, either in uncompressed (e.g., `*.lp`, `*.mps`) or compressed (e.g., `*.lp.gz`, `*.mps.zip`) form.

- Gurobi, Python: use `read()` function
- CPLEX, DOcplex: use `ModelReader` helper class
- CPLEX, legacy Python: construct `Cplex` instance directly from filename
- Gurobi, C++: construct `GRBModel` instance directly from filename
- CPLEX, C++: use `importModel()` function of `IloCplex` instance to read model into an initialized `IloModel` instance

TODO: Links to Relevant Documentation
TODO: Code Examples
TODO: Reading in parameter data from a file

4.3 Modifying the Model

All four APIs have some type of "model" object, which represents the problem at the modeling level (i.e., variables, constraints, objective function). Since the *model* is built incrementally, it is straightforward to modify this representation depending on different parameters and arguments.

For example, suppose the number of constraints you generate depends on some parameter; in this case, you can create a loop whose number of iterations depends on the parameter. For another example, suppose you want to enable/disable a particular set of constraints based on some condition; in this case, you can put the code that adds these constraints inside an if statement.

TODO: Specific Commands
TODO: Links to Relevant Documentation
TODO: Code Examples

4.4 Modifying Solver Parameters

While MIP solvers provide the means to make more significant changes to the algorithm, here we talk about some of the simpler changes.

Solver Limits refer to various solver parameters that limit the amount of system resources devoted to solving the problem. For example, if you are running many experiments you may want to limit the amount of time devoted to solving any one instance. Both Gurobi and CPLEX have various solver parameters that limit the amount of wall clock/system time used by the solver.

For another example, if you are running experiments on a shared server, it is generally recommended that no one user uses all of the system's CPU cores. We can limit the number of CPU cores used by the solver by modifying the `threads` parameter.

Presolve is a crucial part of modern MIP solvers; while many different procedures can be part of a presolve procedure, the overall effect they have can be summarized as *taking the original problem and transforming it into a simpler problem*, whose optimal solution easily yields an optimal solution to the original problem.

For users who are only interested in the solution of the problem, presolve is just another feature that improves solver performance. However, for users who want to analyze the relationship between the model (formulation) and the algorithm performance/behavior, presolve can obscure the relationship being studied since it changes the problem being solved. In this case, it may be desirable to disable presolve.

Numerical precision/tolerance in general refers to how solvers handle the fact that computers do not represent real numbers to arbitrary accuracy by default. This matters, for example, when enforcing an integrality constraint; if the integer variable x's current value is 4.0000000000000001, it is hard to objectively determine whether the "error" is due to loss of precision or because the solution genuinely does not satisfy integrality.

TODO: Specific Commands
TODO: Links to Relevant Documentation
TODO: Code Examples

4.5 Command Line Arguments

Simple command line arguments `sys.argv` in python and `int argc char* [] argv` arguments in C++. For more sophisticated argument parsing, it's helpful to use a library. For Python, the standard module `argparse` is helpful (see https://docs.python.org/3/library/argparse.html for details). For C++, there are no official options but https://github.com/Taywee/args is a header-only library that mimics the syntax of Python's `argparse`.

TODO: Code Examples