Module 3

Ryo Kimura

April 3, 2019

3 Basic Modeling

In this module we will briefly review the fundamentals of MIP models, and provide examples of how to implement them using Gurobi and CPLEX.

NOTE From this point forward, we will assume that your primary programming environment is **Ubuntu Linux**; if you are using a different programming environment, you will need to determine what changes are necessary in order to make it run. If you have trouble, you can contact the instructor for advice on what changes to make.

On the other hand, we will still cover how to model things using both Gurobi and CPLEX, via both Python and C++.

3.1 IP Review

A mixed integer programming (MIP) model is defined by the following characteristics:

- Linear objective function
- Affine inequality constraints
- Integrality constraints

A simple example of a MIP model is the *0-1 knapsack problem*. We have n different objects, each with a *value* c_j and a *weight* a_j, and a knapsack with total capacity b. Our goal is to determine which objects to put in our knapsack so that we maximize their total value, subject to the constraint that the total weight of our objects does not exceed the capacity of the knapsack. Let $x_j = 1$ if object j is included in our knapsack and 0 otherwise. Then we can represent our problem as:

$$\begin{array}{lrcll} \max\limits_{x} & \sum\limits_{j=1}^{n} c_j x_j & & & \\ \text{s.t.} & \sum\limits_{j=1}^{n} a_j x_j & \leq & b & \\ & x_j & \in & \{0,1\} & \forall j = 1,\ldots,n \end{array}$$

where the domain constraint $x_j \in \{0,1\}$ is implicitly represented as the combination of the affine inequality constraints $0 \leq x_j \leq 1$ and the integrality constraint $x_j \in \mathbb{Z}$.

A more involved example of a MIP model is the following *single machine scheduling problem with weighted tardiness objective*. We have n jobs that need to be run on some machine. Each job has the following attributes:

1. **weight** w_j: how important the job is,
2. **duration** p_j: how long it takes to execute the job, and
3. **deadline** d_j: the time by which we want to complete the job.

Thus, if we start executing job j at **start time** S_j, we will finish executing the job at its **completion time** $C_j := S_j + p_j$. In this case, the **tardiness** of job j is defined to be

$$T_j := \max\{C_j - d_j, 0\},$$

i.e., it is 0 if job j finishes on time, and otherwise is the difference between the completion time and deadline of job j. Our goal is to minimize the weighted sum of tardiness over all jobs (i.e., $\min \sum_{j\in\mathcal{J}} w_j T_j$), subject to the constraints that we can only run one job at a time (since we only have one machine), and the earliest we can start running jobs is at time 0.

We can model this problem as a MIP as follows: let $x_{ij} = 1$ if job i is executed before job j and 0 if job j is executed before job i. Let $M := \sum_j p_j$ so that $C_j - M \leq 0$ for any reasonable completion time for any job j. Then we can represent our problem as

$$\begin{array}{llcll}
\min\limits_{S,T,x} & \displaystyle\sum_{j\in\mathcal{J}} w_j T_j & & & \\
\text{s.t.} & S_j & \geq & S_i + p_i - M(1 - x_{ij}) & \forall (i,j) \in \text{Pairs}(\mathcal{J}) \\
 & S_i & \geq & S_j + p_j - M x_{ij} & \forall (i,j) \in \text{Pairs}(\mathcal{J}) \\
 & T_j & \geq & S_j + p_j - d_j & \forall j \in \mathcal{J} \\
 & x_{ij} & \in & \{0,1\} & \forall (i,j) \in \text{Pairs}(\mathcal{J}) \\
 & S_j, T_j & \geq & 0 & \forall j \in \mathcal{J}
\end{array}$$

where $\mathcal{J} = \{1, \ldots, n\}$ is the set of n jobs and $\text{Pairs}(\mathcal{J}) = \{(i,j) \in \mathcal{J} \times \mathcal{J} \mid i < j\}$ is the set of all pairs of jobs in $\mathcal{J}$.

It is helpful to think of a MIP model as being made up of four elements: *parameters*, *variables*, the *objective function*, and *constraints*. In the previous model, these elements were:

- *Parameters*: w_j, p_j, d_j, which are defined in the problem, and M, a *big-M* parameter calculated based on p_j; note that parameters are **constant w.r.t. the variables** (e.g., the value of M does not depend on the value of x_{ij}, S_j, or T_j)
- *Variables*: x_{ij}, which are *integer* (in particular, *binary*) variables, and S_j and T_j, which are (nonnegative) *continuous* variables
- *Objective Function*: $\sum_{j\in\mathcal{J}} w_j T_j$, which is a linear function in the variable T_j
- *Constraints*: everything else, e.g., $S_j \geq S_i + p_i - M(1 - x_{ij})$, which is an inequality constraint involving the variables S_i, S_j, and x_{ij}

Modern MIP solvers like Gurobi and CPLEX use a combination of *branch-and-bound*, general purpose *cutting planes*, *primal heuristics*, and *presolve reductions*, among other things, to solve MIPs. Later in the course, we will discuss how to manipulate these solver features; for now, it suffices to note that modern MIP solvers can often find the optimal solutions to MIPs of moderate size in a reasonable amount of time.

3.2 Gurobi Example (Python)

We will use Example 3.6.3 from [1], which uses the last MIP model we discussed, as our running example. We can use Gurobi's Python API to implement this MIP model. The basic structure of such a program consists of the following steps:

1. **Specify which interpreter to use** (line 1): Specify the path to the Python interpreter that will be used to run this program.

2. **Import the gurobipy package** (line 2): Import functions from the `gurobipy` package we will use to model our problem.

3. **Set up the problem parameters** (lines 6-12): Defining and populating data structures containing parameter data makes it easier to refer to them later on when we specify the model. *Lists* and *dicts* are especially recommended due to their flexibility.

4. **Set up the model** (lines 15-37): Create a `Model` object and add variables, constraints, and an objective function to it.

5. **Solve the model** (lines 39-44): Use `model.optimize()` to solve the model, and re-solve it if we end with ambiguous solver status.

6. **Display the solution** (lines 46-53): Report optimal solution if one was found

In addition, our program also features:

1. **Status descriptions** (line 3): Define a dict mapping status enums (i.e., integers) to more human-readable strings.

2. **Script function** (lines 5, 58, 59): Define primary code as a function, then call it from "main" function. This makes it easier to test/parameterize our code later on.

3. **Exception Handling** (lines 14, 55, 56): Use a `try-catch` block to handle solver errors.

Here is the full Python code for our model. The program can also be found on Canvas with some of the lines adjusted. See the `examples/python` directory of the Gurobi distribution for more Python examples distributed by Gurobi, and see http://www.gurobi.com/documentation/8.1/refman/py_python_api_details.html for details on the Python API.

```
1  #!/usr/bin/python3
2  from gurobipy import Model, GRB, GurobiError, quicksum
3  StatusDict = {getattr(GRB.Status, s): s for s in dir(GRB.Status) if s.isupper()}
4
5  def TWTgurobi():
6      # TWT Problem Data
7      jobs = tuple([i+1 for i in range(4)])
8      jobPairs = [(i,j) for i in jobs for j in jobs if i < j]
9      weight = dict(zip(jobs, (4, 5, 3, 5)))
10     duration = dict(zip(jobs, (12, 8, 15, 9)))
11     deadline = dict(zip(jobs, (16, 26, 25, 27)))
12     M = sum(duration.values())
13
14     try:
```

```python
        # Create a new model
        m = Model('TWTexample')

        # Create variables
        # x[(i,j)] = 1 if i << j, else j >> i
        x = m.addVars(jobPairs, vtype=GRB.BINARY, name='x')
        startTime = m.addVars(jobs, name='startTime')
        tardiness = m.addVars(jobs, name='tardiness')

        # Set objective function
        m.setObjective(quicksum([weight[j]*tardiness[j] for j in jobs]),
            GRB.MINIMIZE)

        # Add constraints
        m.addConstrs(
            (startTime[j] >= startTime[i] + duration[i] - M*(1-x[(i,j)])
                for (i,j) in jobPairs), 'NoOverlap1')
        m.addConstrs(
            (startTime[i] >= startTime[j] + duration[j] - M*x[(i,j)]
                for (i,j) in jobPairs), 'NoOverlap2')
        m.addConstrs(
            (tardiness[j] >= startTime[j] + duration[j] - deadline[j]
                for j in jobs), 'Deadline')

        # Solve model
        m.optimize()
        if m.status == GRB.Status.INF_OR_UNBD:
            # Disable dual reductions to determine solve status
            m.setParam(GRB.Param.DualReductions, 0)
            m.optimize()

        # Display solution
        if m.status == GRB.Status.OPTIMAL:
            for v in m.getVars():
                print('%s:\t%g' % (v.varName, v.x))
            print('Objective:\t%g' % m.objVal)
        else:
            statstr = StatusDict[m.status]
            print('Optimization was stopped with status %s' % statstr)

    except GurobiError as e:
        print('Error code ' + str(e.errno) + ": " + str(e))

if __name__ == '__main__':
    TWTgurobi()
```

3.3 CPLEX Example (Python)

We will implement the same example using CPLEX's DOcplex Python API. The basic structure of this program is similar to our last model:

1. **Specify which interpreter to use** (line 1): Specify the path to the Python interpreter that will be used to run this program.

2. **Import the docplex/docloud package** (lines 2-3): Import functions from the `docplex` and `docloud` packages we will use to model our problem.

3. **Set up the problem parameters** (lines 6-12): Defining and populating data structures containing parameter data makes it easier to refer to them later on when we specify the model. *Lists* and *dicts* are especially recommended due to their flexibility.

4. **Set up the model** (lines 15-34): Create a `Model` object and add variables, constraints, and an objective function to it.

5. **Solve the model** (lines 36-42): Use `model.solve()` to solve the model, and re-solve it if we end with ambiguous solver status.

6. **Display the solution** (lines 44-51): Report optimal solution if one was found

In addition, our program also features:

1. **Context Management** (line 15): using `with` means that the `Model` object is automatically deleted when its corresponding code block finishes execution

2. **Script function** (lines 5,53,54): Define primary code as a function, then call it from "main" function. This makes it easier to test/parameterize our code later on.

Here is the full Python code for our model. The program can also be found on Canvas with some of the lines adjusted. See the `python/examples/mp/modeling` directory of the CPLEX distribution for more Python examples distributed by CPLEX, and see http://ibmdecisionoptimization.github.io/docplex-doc/mp/refman.html for details on the DOcplex API.

```
1  #!/usr/bin/python3
2  from docplex.mp.model import Model
3  from docloud.status import JobSolveStatus
4
5  def TWTdocplex():
6      # TWT Problem Data
7      jobs = tuple([i+1 for i in range(4)])
8      jobPairs = [(i,j) for i in jobs for j in jobs if i < j]
9      weight = dict(zip(jobs, (4, 5, 3, 5)))
10     duration = dict(zip(jobs, (12, 8, 15, 9)))
11     deadline = dict(zip(jobs, (16, 26, 25, 27)))
12     M = sum(duration.values())
13
14     # Create a new model
15     with Model(name='TWTexample', log_output=True) as m:
16         # Create variables
17         # x_i_j = 1 if i << j, else j >> i
18         x = m.binary_var_dict(jobPairs, name='x')
19         startTime = m.continuous_var_dict(jobs, name='startTime')
20         tardiness = m.continuous_var_dict(jobs, name='tardiness')
21
```

```python
        # Set objective function
        m.minimize(m.sum(weight[j]*tardiness[j] for j in jobs))

        # Add constraints
        m.add_constraints(
            (startTime[j] >= startTime[i] + duration[i] - M*(1-x[(i,j)]),
                'NoOverlap1_%d_%d' % (i,j)) for (i,j) in jobPairs)
        m.add_constraints(
            (startTime[i] >= startTime[j] + duration[j] - M*x[(i,j)],
                'NoOverlap2_%d_%d' % (i,j)) for (i,j) in jobPairs)
        m.add_constraints(
            (tardiness[j] >= startTime[j] + duration[j] - deadline[j],
                'Deadline_%d' % j) for j in jobs)

        # Solve model
        m.solve()
        mstatus = m.get_solve_status()
        if mstatus == JobSolveStatus.INFEASIBLE_OR_UNBOUNDED_SOLUTION:
            # Disable primal/dual reductions to determine solve status
            m.parameters.preprocessing.reduce = False
            mstatus = m.get_solve_status()

        # Display solution
        if mstatus == JobSolveStatus.OPTIMAL_SOLUTION:
            for v in m.iter_variables():
                print('%s:\t%g' % (v.name, v.solution_value))
            print('Objective:\t%g' % m.objective_value)
        else:
            statstr = m.solve_details.status
            print('Optimization was stopped with status %s' % str(statstr))

if __name__ == '__main__':
    TWTdocplex()
```

3.4 Gurobi Example (C++)

We will implement the same example using Gurobi's C++ API. The basic structure is similar:

1. **Include relevant libraries** (lines 1-7): Specify libraries that we will use, including the Gurobi C++ API.

2. **Set up the problem parameters** (lines 15-24): We primarily use `const std::array` due to its standardized design.

3. **Set up the model** (lines 26-93): Construct the `Model` object. We use `std::ostringstream` throughout to construct the name strings.

   (a) **Create model** (lines 26-29)

   (b) **Add variables** (lines 31-61): We use `emplace`, `reserve`, and `emplace_back` to efficiently store variables that can later be retrieved by index.

(c) **Set objective function** (lines 63-69)

(d) **Add constraints** (lines 71-93): We use a range-based for loop to iterate over the elements of `GRBVarPairMap`.

4. **Solve the model** (lines 95-101): Use `model.optimize()` to solve the model, and re-solve it if we end with ambiguous solver status.

5. **Display the solution** (lines 103-120): Report optimal solution if one was found, and solver status otherwise

In addition, our program also features:

1. **Typedefs** (lines 8-9): We use *typedefs* to assign convenient names to our template classes.

   (a) **Map Template** (line 8): We use the `std::map` template to define a data structure that uses a key-value representation rather than a matrix-based reprsentation.

   (b) **Smart Pointer** (line 9): We use the `std::unique_ptr` template to automatically handle the dynamic memory allocation of `GRBVar[]`

2. **Exception Handling** (lines 13-14, 121-130): Use a `try-catch` block to handle solver errors.

Here is the full C++ code for our model. The program can also be found on Canvas with some of the lines adjusted. See the `examples/cpp` directory of the Gurobi distribution for more C++ examples distributed by Gurobi, and see http://www.gurobi.com/documentation/current/refman/cpp_api_details.html for details on the C++ API.

```
1  #include <tuple>
2  #include <map>
3  #include <memory>
4  #include <array>
5  #include <vector>
6  #include <sstream>
7  #include "gurobi_c++.h"
8  typedef std::map<std::tuple<int,int>,GRBVar> GRBVarPairMap;
9  typedef std::unique_ptr<GRBVar[]> GRBVarArray;
10
11 int main(int argc, char* argv[])
12 {
13     try
14     {
15         // TWT Problem Data
16         const int nbJobs = 4;
17         const std::array<double, nbJobs> weight = {4, 5, 3, 5};
18         const std::array<double, nbJobs> duration = {12, 8, 15, 9};
19         const std::array<double, nbJobs> deadline = {16, 26, 25, 27};
20         double M = 0;
21         for (int j = 0; j < nbJobs; ++j)
22         {
23             M += duration[j];
24         }
25
```

```
        // Create a new model
        GRBEnv env = GRBEnv();
        GRBModel model = GRBModel(env);
        model.set(GRB_StringAttr_ModelName, "TWTexample");

        // Create variables
        // x(i)(j) = 1 if i << j, else j >> i
        GRBVarPairMap x;
        for (int i = 0; i < nbJobs; ++i)
        {
            for (int j = i+1; j < nbJobs; ++j)
            {
                std::ostringstream varname;
                varname << "x(" << i << ")(" << j << ")";
                x.emplace(std::make_tuple(i, j),
                    model.addVar(0.0, 1.0, 0.0, GRB_BINARY, varname.str()));
            }
        }
        std::vector<GRBVar> startTime;
        startTime.reserve(nbJobs);
        for (int j = 0; j < nbJobs; ++j)
        {
            std::ostringstream varname;
            varname << "startTime(" << j << ")";
            startTime.emplace_back(
                model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, varname.str()));
        }
        std::vector<GRBVar> tardiness;
        tardiness.reserve(nbJobs);
        for (int j = 0; j < nbJobs; ++j)
        {
            std::ostringstream varname;
            varname << "tardiness(" << j << ")";
            tardiness.emplace_back(
                model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, varname.str()));
        }

        // Set objective function
        GRBLinExpr obj = 0;
        for (int j = 0; j  < nbJobs; ++j)
        {
            obj += weight[j]*tardiness[j];
        }
        model.setObjective(obj, GRB_MINIMIZE);

        // Add constraints
        for (auto& kv : x)
        {
            int i, j;
            GRBVar x_ij;
```

```
            std::forward_as_tuple(std::tie(i, j), x_ij) = kv;

            std::ostringstream consname;
            consname << "NoOverlap1_" << i << "_"  << j;
            model.addConstr(startTime[j] >= startTime[i] + duration[i] - M*(1-x_ij),
                consname.str());
            consname.str("");
            consname << "NoOverlap2_" << i << "_" << j;
            model.addConstr(startTime[i] >= startTime[j] + duration[j] - M*x_ij,
                consname.str());
        }
        for (int i = 0; i < nbJobs; ++i)
        {
            std::ostringstream consname;
            consname << "Deadline_" << i;
            model.addConstr(tardiness[i] >= startTime[i] + duration[i] - deadline[i],
                consname.str());
        }

        // Solve model
        model.optimize();
        if (model.get(GRB_IntAttr_Status) == GRB_INF_OR_UNBD)
        {
            model.set(GRB_IntParam_DualReductions, 0);
            model.optimize();
        }

        // Display solution
        if (model.get(GRB_IntAttr_Status) == GRB_OPTIMAL)
        {
            int numVars = model.get(GRB_IntAttr_NumVars);
            auto varArray = GRBVarArray(model.getVars());
            for (int i = 0; i < numVars; ++i)
            {
                std::string varname = varArray[i].get(GRB_StringAttr_VarName);
                double varval = varArray[i].get(GRB_DoubleAttr_X);
                std::cout << varname << ":\t" << varval << std::endl;
            }
            std::cout << "Objective:\t" << model.get(GRB_DoubleAttr_ObjVal) << std::endl;
        }
        else
        {
            int status = model.get(GRB_IntAttr_Status);
            std::cout << "Optimization was stopped with status " << status << std::endl;
        }
    }
    catch (const GRBException& e)
    {
        std::cout << "Error code = " << e.getErrorCode() << std::endl;
        std::cout << e.getMessage() << std::endl;
```

```
    }
    catch (...)
    {
        std::cout << "Exception during optimization" << std::endl;
    }
    return 0;
}
```

3.5 CPLEX Example (C++)

We will implement the same example using CPLEX's C++ API. The basic structure is similar:

1. **Include relevant libraries** (lines 1-4): Specify libraries that we will use, including the CPLEX C++ API.

2. **Initialize environment** (line 9): The `IloEnv` object provides a platform-independent context for everything we do.

3. **Set up the problem parameters** (lines 12-17): Use CPLEX's types (`IloInt` ≡ `int`, `IloNumArray` ≡ `std::array<double,N>`) for more platform-independent code.

4. **Set up the model** (lines 19-68): Construct the `IloModel` object. Use `std::ostringstream` throughout to construct the name strings.

   (a) **Create model** (lines 19-20)

   (b) **Add variables** (lines 22-37): Use `emplace` to store variables in `IloBoolVarPairMap` that can later be retrieved by index.

   (c) **Set objective function** (lines 39-40)

   (d) **Add constraints** (lines 42-68): Use a range-based for loop to iterate over the elements of `GRBVarPairMap`.

5. **Solve the model** (lines 70-77): Use `cplex.solve()` to solve the model, and re-solve it if we end with ambiguous solver status.

6. **Display the solution** (lines 79-99): Report optimal solution if one was found, and solver status otherwise

7. **End environment** (line 108)

In addition, our program also features:

1. **Typedef/Map Template** (line 5): Use *typedefs* to assign a convenient name to the `std::map` template, which defines a data structure that uses a key-value representation rather than a matrix-based reprsentation.

2. **Exception Handling** (lines 10-11, 99-107): Use a `try-catch` block to handle solver errors.

Here is the full C++ code for our model. The program can also be found on Canvas with some of the lines adjusted. See the `cplex/examples/src/cpp` directory of the CPLEX distribution for more C++ examples distributed by Gurobi, and see https://www.ibm.com/support/knowledgecenter/SSSA5P_latest/ilog.odms.cplex.help/refcppcplex/html/overview.html?pos=2 for details on the C++ API.

```
#include <tuple>
#include <map>
#include <sstream>
#include <ilcplex/ilocplex.h>
typedef std::map<std::tuple<IloInt,IloInt>,IloBoolVar> IloBoolVarPairMap;

int main(int argc, char* argv[])
{
    IloEnv env;
    try
    {
        // TWT Problem Data
        const IloInt nbJobs = 4;
        const IloNumArray weight(env, nbJobs, 4, 5, 3, 5);
        const IloNumArray duration(env, nbJobs, 12, 8, 15, 9);
        const IloNumArray deadline(env, nbJobs, 16, 26, 25, 27);
        const IloNum M = IloSum(duration);

        // Create a new model
        IloModel model(env, "TWTexample");

        // Create variables
        // x(i)(j) = 1 if i << j, else j >> i
        IloBoolVarPairMap x;
        for (IloInt i = 0; i < nbJobs; ++i)
        {
            for (IloInt j = i+1; j < nbJobs; ++j)
            {
                std::ostringstream varname;
                varname << "x(" << i << ")(" << j << ")";
                x.emplace(std::make_tuple(i, j), IloBoolVar(env, varname.str().c_str()));
            }
        }
        IloNumVarArray startTime(env, nbJobs, 0.0, IloInfinity);
        startTime.setNames("startTime");
        IloNumVarArray tardiness(env, nbJobs, 0.0, IloInfinity);
        tardiness.setNames("tardiness");

        // Set objective function
        model.add(IloMinimize(env, IloScalProd(weight, tardiness) ));

        // Add constraints
        for (auto& kv : x)
        {
            IloInt i, j;
            IloBoolVar x_ij;
            std::forward_as_tuple(std::tie(i, j), x_ij) = kv;

            IloConstraint ovCons1(startTime[j] >= startTime[i] + duration[i] - M*(1-x_ij));
            std::ostringstream consname;
```

```
            consname << "NoOverlap1_" << i << "_"  << j;
            ovCons1.setName(consname.str().c_str());
            model.add(ovCons1);

            IloConstraint ovCons2(startTime[i] >= startTime[j] + duration[j] - M*x_ij);
            consname.str("");
            consname << "NoOverlap2_" << i << "_"  << j;
            ovCons2.setName(consname.str().c_str());
            model.add(ovCons2);
        }
        for (IloInt i = 0; i < nbJobs; i++)
        {
            IloConstraint ddlCons(tardiness[i] >= startTime[i] + duration[i] - deadline[i]);
            std::ostringstream consname;
            consname << "Deadline_" << i;
            ddlCons.setName(consname.str().c_str());
            model.add(ddlCons);
        }

        // Solve model
        IloCplex cplex(model);
        cplex.solve();
        if (cplex.getStatus() == IloAlgorithm::InfeasibleOrUnbounded)
        {
            cplex.setParam(IloCplex::Param::Preprocessing::Reduce, 0);
            cplex.solve();
        }

        // Display solution
        if (cplex.getStatus() == IloAlgorithm::Optimal)
        {
            for (IloIterator<IloNumVar> it(env); it.ok(); ++it)
            {
                IloNumVar var = *it;
                if (cplex.isExtracted(var))
                {
                    std::string varname = var.getName();
                    double varval = cplex.getValue(var);
                    env.out() << varname << ":\t" << varval << std::endl;
                }
            }
            env.out() << "Objective:\t" << cplex.getObjValue() << std::endl;
        }
        else
        {
            IloAlgorithm::Status status = cplex.getStatus();
            env.out() << "Optimization was stopped with status " << status << std::endl;
        }
    }
    catch (const IloException& ex)
```

```
    {
        std::cerr << "CPLEX Error: " << ex << std::endl;
    }
    catch (...)
    {
        std::cerr << "Error" << std::endl;
    }
    env.end();
    return 0;
}
```

3.6 Optional Content

3.6.1 Imports in Python

Imports in Python, while very flexible, can be the source of frustratingly obtuse errors. This section is meant to clarify some of the confusing elements of the Python import system. It's based on *The Definitive Guide to Python import Statements* (https://chrisyeh96.github.io/2017/08/08/definitive-guide-python-imports.html) by Chris Yeh.

Modules, Packages, and Objects First, it's useful to clarify the distinction between a *module*, a *package*, and a *object*:

- A *module* is any `*.py` file. Its name is the file name.
- A *package* is any folder containing a file named `__init__.py` in it. Its name is the name of the folder.
  - In Python 3.3 and above, any folder (even without a `__init__.py` file) is considered a package
- An *object* is pretty much anything in Python - functions, classes ,variables, etc.

Thus, a *module* is a Python file that implements various *objects*, while a *package* is conceptually a collection of *modules* bundled together for distribution. By default, an object defined in a module cannot be directly accessed from the package containing that module, unless the creator of the package makes it available via the package's `__init__.py` file.

Where does Python look for modules/packages? When a module named `spam` is imported, Python first searches for a *built-in module* (i.e., a module that are compiled directly into the Python interpreter) with that name. If not found, it then searches for a file (i.e., module) named `spam.py` or a folder (i.e., package) named `spam` in a list of directories given by the variable `sys.path`. `sys.path` is initialized from various locations, including the directory containing the input script, `PYTHONPATH`, and the installation-dependent default.

Using Objects from the Imported Module or Package There are 4 different syntaxes for writing import statements. (Note that importing a package is conceptually the same as importing that package's `__init__.py` file.)

1. `import <package>`

2. import <module>
3. from <package> import <module or subpackage or object>
4. from <module> import <object>

Let X be whatever name comes after import:

- If X is the name of a module or package, then to use objects defined in X, you have to write X.object.
- If X is a variable name, then it can be used directly.
- If X is a function name, then it can be invoked with X().

Optionally, as Y can be added after any import X statement: import X as Y. This renames X to Y within the script. Note that the name X itself is no longer valid. A common example is import numpy as np.

Examples For Gurobi, gurobipy is a package which contains the object Model.

- **Method 1**: from gurobipy import Model
  - we can directly use the object by name: m = Model()
- **Method 2**: from gurobipy import *
  - we can directly use all objects defined in gurobipy by name; need to watch out for naming conflicts (e.g., you cannot have a variable called Model in your script!)
- **Method 3**: import gurobipy as grb
  - we have to prefix the object name with the (re-named) name of the module: m = grb.Model()

For CPLEX, docplex is a package, which contains the subpackage mp, which contains the module model which contains the object Model.

- **Method 1**: from docplex.mp.model import Model
  - we can directly use the object by name: m = Model()
- **Method 2**: from docplex.mp import model or equivalently import docplex.mp.model as model
  - we have to prefix the function name with the name of the module: m = model.Model()
  - This is sometimes preferred over the Method 1 in order to make it explicit that we are using the Model object from the model module.
- **Method 3**: import docplex.mp.model
  - we need to use the full path: m = docplex.mp.model.Model()

References

[1] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems.* 4th ed. Springer, 2012.