how_to_mip

How to MIP

What is **MIP**?

MIP is a simple Python package that can be used to solve optimization problems.

To solve a problem, you need to follow a series of steps:

1. Create your model.
2. Define the decision variables.
3. Set the objective function.
4. Add constraints.
5. Solve the problem.

We will now see how to tackle each part in detail.

1. Create your Model

To create your model, you need to call the `Model` constructor from the `mip` package.

```
model = mip.Model()
```

---

2. Define the Decision Variable

To define the decision variables, you need to use the `add_var` method.

Let's look at different examples and situations.

1. **Defining a simple list of variables**

   ```
   x = [model.add_var(name=i, lb=0) for i in I]
   ```

   Here, for every `i` in the list `I`, we assign a variable. The `name` option allows us to set the variable's name, `lb` sets the lower bound, and you can also use other parameters like `ub` to set the upper bound.

   The result here is a list of variables with initial values of `0` that can take fractional values.

---

2. **One Step Forward**

   Now, let's imagine we want to create variables with the following definitions:

   - x_{ij}: Units of oil of type $i \in I$ used for gasoline of type $j \in J$

   The code looks like this:

```
x = {(i, j): model.add_var(name="name", lb=0) for i in I for j in J}
```

---

3. **Integer Variables**

   If you want your variables to take integer values, you can use the `var_type` parameter of the `add_var` method.

```
x = {(i, j): model.add_var(name="name", var_type=INTEGER, lb=0) for i in
I for j in J}
```

   There are two ways to use `INTEGER` as `var_type`:

```
# Import it and use it directly like the example above
from mip import INTEGER

# Use it by dot notation
var_type = mip.INTEGER
```

---

4. **Binary Variables**

   It's similar to defining integer variables; however, instead of using `INTEGER` as `var_type`, we use `BINARY`.

```
x = {(i, j): model.add_var(name="name", var_type=BINARY) for i in I for
j in J}
```

---

3. Set the Objective Function

To set the objective function, you can use the `minimize()` or `maximize()` methods, and you must assign it to the `objective` property of the model.

```
model.objective = mip.maximize()

model.objective = mip.minimize()
```

Since the objective function is linear and involves summation, you should use `xsum` to create the same functionality. Let's convert the following objective function to code:

$$\max \sum_{i \in I, j \in J} p_j x_{ij}$$

```
model.objective = mip.maximize(mip.xsum(p[j] * x[i, j] for i in I for j in
J))
```

---

4. Add Constraints

Constraints are used to limit the possible values of variables. They are added using the `add_constr` method. You can define constraints using the following structure:

```
model.add_constr(left_hand_side comparison_sign right_hand_side)
```

For example, let's write the code for the following constraint:

$$\sum_{j \in J} x_{ij} \leq b_i \quad \forall i \in I$$

```
for i in I:
    model.add_constr(mip.xsum(x[i, j] for j in J) <= b[i])
```

---

5. Solve the Problem

After defining all constraints, you can solve the problem by calling the `optimize()` method. If you receive `0` as the output, it means the problem is solved. Otherwise, receiving `1` indicates the problem is infeasible or unbounded, and no solution is generated.

```
# To solve the model
model.optimize()

# To see the objective value
```

```python
objective_value = model.objective.x

# To see the values of variables
for var in model.vars:
    print(f"{var.name}: {var.x}")
```