Warehouse location with conflict

Warehouse location with conflict

This example is a variant of the warehouse location problem that illustrates the use of two CPLEX features: conflict refiner and FeasOpt.

Models can be infeasible for various reasons, including incompatible constraints and bounds or incompatible data. CPLEX can identify conflicting constraints and bounds in a model. The conflict refiner examines elements within an infeasible model that can be removed from the conflict in order to arrive at a minimal conflict. This minimal conflict can make it easier to identify the source of infeasibilities in the original model.

The FeasOpt feature attempts to repair an infeasible model according to preferences you set. FeasOpt selectively relaxes bounds and constraints in a way that minimizes a user-defined weighted penalty function. FeasOpt tries to suggest the least change to your model that would achieve feasibility. It does not modify the model itself.

Infeasible LP model

This example is a variant of the warehouse location example. To illustrate the conflict refiner and FeasOpt functionality of DOcplexcloud, the capacities of all warehouses have been reduced to serve only one store and the model has been transformed from `.mod` to `.lp` format. The infeasible LP model is used in four examples to show how the various algorithms are configured and how they work.

- All four algorithms take an XML input file defined by a `feasibility.xsd` schema.
- The two variants of the conflict refiner produce a `conflict.xml` file or `conflict.json` file, depending on the selected output type. The format of these two files is specified by the schemas `conflict.xsd` and `conflictschema.json` available to download from the DOcplexcloud DevCenter.
- The two variants of FeasOpt return a `solution.xml` or `solution.json` file, depending on the selected output type. This is the same solution file format as that produced by a regular solve process. Its schema is available to download from the DOcplexcloud DevCenter.
- In a `.feasibility` file, variables and constraints can be referenced by index or by name. Referencing by index should be preferred because it is a lot faster than referencing by name. Care is necessary when referencing variables by name and using LP files: When reading an LP file, the ordering of variables in CPLEX is the order in which the variables appear in the LP file.

Conflict refiner variant: `conflict` directory

DOcplexcloud identifies the model in `conflict/warehouse_infeasible.lp` as infeasible. To invoke the conflict refiner, submit the file `conflict/warehouse_infeasible_conflict.feasibility` at the same time as the the LP file. This file contains a single entry:

```
<CPLEXRefineconflict/>
```

This instructs CPLEX to use the conflict refiner on the accompanying LP model.

DOcplexcloud runs the conflict refiner and returns an XML or JSON file containing details of the conflict. For example in XML:

```
<?xml version='1.0' encoding='UTF-8'?>
<CPLEXConflict>
  <row index='0' status='member'/>
  <row index='1' status='member'/>
  <row index='2' status='member'/>
  <row index='3' status='member'/>
  <row index='4' status='member'/>
  <row index='5' status='member'/>
  <row index='60' status='member'/>
  <row index='61' status='member'/>
  <row index='62' status='member'/>
  <row index='63' status='member'/>
  <row index='64' status='member'/>
</CPLEXConflict>
```

The conflict refiner identifies that the constraints with the following indices are in conflict:

```
  0 : ctEachStoreHasOneWarehouse(0)
  1 : ctEachStoreHasOneWarehouse(1)
  2 : ctEachStoreHasOneWarehouse(2)
  3 : ctEachStoreHasOneWarehouse(3)
  4 : ctEachStoreHasOneWarehouse(4)
  5 : ctEachStoreHasOneWarehouse(5)
 60 : ctMaxUseOfWarehouse("Bonn")
 61 : ctMaxUseOfWarehouse("Bordeaux")
 62 : ctMaxUseOfWarehouse("London")
 63 : ctMaxUseOfWarehouse("Paris")
 64 : ctMaxUseOfWarehouse("Rome")
```

As expected, the warehouses do not provide enough capacity to meet all demand. The set of constraints returned by the conflict refiner is clearly in conflict:

- The `ctEachStoreHasOneWarehouse` constraints require the opening of six stores.
- There are only five warehouses.
- The `ctMaxUseOfWarehouse` constraints limit the number of stores served by a warehouse to one.

Thus, it's possible to accommodate for only five warehouses and the set of constraints is in conflict.

The conflict is also minimal because removing any of the constraints from the conflict will result in a feasible set of constraints:

- If you remove one of the `ctEachStoreHasOneWarehouse` constraints, you only need to open five stores. The warehouses have enough capacity to serve five stores.
- If you remove one of the `ctMaxUseOfWarehouse` constraints, the respective warehouse has infinite capacity and it's possible to accommodate for all the stores.

Note that the conflict refiner will only consider linear constraints and bounds of variable for a potential conflict. If you have other constraints (indicator, SOS, quadratic constraints) you should use the extended conflict refiner. See the CPLEX documentation for more details.

Extended conflict refiner variant: `conflictext` directory

This example shows the use of the extended conflict refiner to examine explicitly the demand requirements and capacity restrictions. The `.feasibility` file defines

two groups of constraints, one for the demand requirements, `ctEachStoreHasOneWarehouse`, and one for the capacity restrictions, `ctMaxUseOfWarehouse`. The constraints are referenced by index in the `warehouse_infeasible_conflictext.feasibility` file:

```
<CPLEXRefineconflictext>
  <group preference="1.0">
    <con index="0" type="lin"/>
    <con index="1" type="lin"/>
    <con index="2" type="lin"/>
    <con index="3" type="lin"/>
    <con index="4" type="lin"/>
    <con index="5" type="lin"/>
    <con index="6" type="lin"/>
    <con index="7" type="lin"/>
    <con index="8" type="lin"/>
    <con index="9" type="lin"/>
  </group>
  <group preference="1.0">
    <con index="60" type="lin"/>
    <con index="61" type="lin"/>
    <con index="62" type="lin"/>
    <con index="63" type="lin"/>
    <con index="64" type="lin"/>
  </group>
</CPLEXRefineconflictext>
```

The extended conflict refiner identifies that these two groups of constraints constitute a minimal conflict. For example in JSON:

```
{
            "CPLEXConflict": {
              "grp": [
                { "index": 0, "status": "member" },
                { "index": 1, "status": "member" }
              ]
            }
          }
```

The constraints in the two groups are clearly in conflict and removing either of the two groups renders the model feasible.

You can use the extended conflict refiner if you already have an idea what the problematic parts of your model might be. Reducing the number of constraints that CPLEX puts into a conflict speeds up the search for a conflict. The extended conflict refiner may also be useful if your model has non-linear constraints: SOS, indicator, or quadratic constraints. See the CPLEX documentation for further details.

FeasOpt variant: `feasopt` directory

This example shows how to run FeasOpt with the infeasible warehouse model. Assuming that you know the infeasibility is coming from the capacity constraints, you can check what is the minimal relaxation of these constraints that will make the model feasible. In the `warehouse_infeasible_feasopt.feasibility` file, you relax the capacity constraints for the five warehouses:

```
<CPLEXFeasopt>
  <rhs>
    <relax index="60" preference="1.0"/> <!-- ctMaxUseOfWarehouse("Bonn") -->
    <relax index="61" preference="2.0"/> <!-- ctMaxUseOfWarehouse("Bordeaux") -->
    <relax index="62" preference="3.0"/> <!-- ctMaxUseOfWarehouse("London") -->
```

```
    <relax index="63" preference="4.0"/> <!-- ctMaxUseOfWarehouse("Paris") -->
    <relax index="64" preference="5.0"/> <!-- ctMaxUseOfWarehouse("Rome") -->
  </rhs>
</CPLEXFeasopt>
```

Relaxing the right-hand side of these constraints means relaxing the capacity limits. The preference specifies your level of willingness to relax a given constraint. The higher the value of the preference, the more you are willing to relax the constraint. With this input FeasOpt could produce this solution:

```
  Supply(0)("Rome")     = 1
  Supply(1)("Paris")"   = 1
  Supply(2)("Rome")"    = 1
  Supply(3)("Bonn")"    = 1
  Supply(4)("Bordeaux") = 1
  Supply(5)("Rome")"    = 1
  Supply(6)("Rome")"    = 1
  Supply(7)("Rome")"    = 1
  Supply(8)("London")"  = 1
  Supply(9)("Rome")"    = 1
```

Each warehouse is opened and supplies at least one store. The excess demand is satisfied by the warehouse in Rome because you specified that warehouse as the one for which you are most willing to relax the capacity constraint.

In addition to relaxing constraints, FeasOpt can relax range values and bounds of variables. See the CPLEX documentation for more details about this feature.

Extended FeasOpt variant: `feasoptext` directory

This is very similar to the FeasOpt variant, except that there is no preference weighting to prioritize which capacity constraint to relax. The capacity constraints are grouped and any of them can be relaxed in order to obtain a feasible model. The `warehouse_infeasible_feasoptext.feasibility` file contains:

```
<CPLEXFeasoptext>
  <group preference="1.0">
    <con index="60" type="lin"/> <!-- ctMaxUseOfWarehouse("Bonn") -->
    <con index="61" type="lin"/> <!-- ctMaxUseOfWarehouse("Bordeaux") -->
    <con index="62" type="lin"/> <!-- ctMaxUseOfWarehouse("London") -->
    <con index="63" type="lin"/> <!-- ctMaxUseOfWarehouse("Paris") -->
    <con index="64" type="lin"/> <!-- ctMaxUseOfWarehouse("Rome") -->
  </group>
</CPLEXFeasoptext>
```

Relaxing any constraint in this group has the same weighting. With this configuration, CPLEX can choose which constraint to relax. It may even choose to relax multiple constraints, all at the same cost.

See the CPLEX documentation for further details about the extended FeasOpt algorithm.