Using COIN-OR to Solve the Uncapacitated Facility Location Problem
An Application Using the Open Solver Interface

Matthew Saltzman

School of Mathematical and Statistical Sciences
Clemson University

COIN fORgery, IMA, October 18, 2018
Outline

1. The Uncapacitated Facility Location Problem
   - UFL Formulation
   - Cutting Planes

2. Developing a Solver
   - The ufl Class
   - COIN Tools
   - Putting It All Together

3. Additional Resources
The following are the input data needed to describe an instance of the uncapacitated facility location problem (UFL):

**Data**
- a set of depots $N = \{1, \ldots, n\}$, a set of clients $M = \{1, \ldots, m\}$,
- the unit transportation cost $c_{ij}$ to service client $i$ from depot $j$,
- the fixed cost $f_j$ for opening depot $j$,
- the demand $d_i$ for client $i$.

**Variables**
- $x_{ij}$ is the amount of the demand for client $i$ satisfied from depot $j$,
- $y_j$ is 1 if the depot is used, 0 otherwise.
The following is a mathematical programming formulation of the UFL:

**UFL Formulation**

\[
\text{Minimize} \quad \sum_{i \in M} \sum_{j \in N} c_{ij}x_{ij} + \sum_{j \in N} f_jy_j \tag{1}
\]

subject to

\[
\sum_{j \in N} x_{ij} = d_i \quad \forall i \in M, \tag{2}
\]

\[
\sum_{i \in M} x_{ij} \leq \left( \sum_{i \in M} d_i \right) y_j \quad \forall j \in N, \tag{3}
\]

\[
y_j \in \{0, 1\} \quad \forall j \in N \tag{4}
\]

\[
0 \leq x_{ij} \leq d_i \quad \forall i \in M, j \in N \tag{5}
\]
The formulation presented on the last slide can be tightened by disaggregating the constraints (3).

\[ x_{ij} - d_jy_j \leq 0, \forall i \in M, j \in N. \]

Rather than adding the inequalities to the initial formulation, we can generate them dynamically.

Given the current LP solution, \( x^*, y^* \), we check whether

\[ x_{ij}^* - d_jy_j^* > \epsilon, \forall i \in M, j \in N. \]

We can also generate inequalities valid for generic MILPs.

If a violation is found, we can iteratively add the constraint to the current LP relaxation.
Here is the basic loop for tightening the initial formulation using the dynamically generated inequalities from the previous slide.

**Solving the LP relaxation**

1. Form the initial LP relaxation and solve it to obtain $(\hat{x}, \hat{y})$.

2. Iterate
   1. Try to generate a valid inequality violated by $(\hat{x}, \hat{y})$. If none are violated, STOP.
   2. Optionally, try to generate an improved feasible solution by rounding $\hat{y}$.
   3. Solve the current LP relaxation of the initial formulation to obtain $(\hat{x}, \hat{y})$.
   4. If $(\hat{x}, \hat{y})$ is feasible, STOP. Otherwise, go to Step 1.
C++ Class

class UFL {
private:
    OsiSolverInterface * si;
    double * trans_cost; // c[i][j] -> c[xindex(i, j)]
    double * fixed_cost; // f[j]
    double * demand; // d[j]
    int M; // number of clients (index on i)
    int N; // number of depots (index in j)
    double total_demand; // sum{j in N} d[j]
    int *integer_vars;

    int xindex(int i, int j) {return i*N + j;}
    int yindex(int j) {return M*N + j;}
};
The Uncapacitated Facility Location Problem
Developing a Solver
Additional Resources

Methods

C++ Class

class UFL {
public:
    UFL(const char* datafile);
    ~UFL();
    void create_initial_model();
    double tighten_initial_model(ostream *os = &cout);
    void solve_model(ostream *os = &cout);
};
Open Solver Interface

- Uniform API for a variety of solvers: currently, CBC, CLP, CPLEX, DyLP, GLPK, Gurobi, Mosek, Soplex, SYMPHONY, the Volume Algorithm, and XPRESS-MP supported to varying degrees.
- Read input from MPS or CPLEX LP files or construct instances using COIN-OR data structures.
- Manipulate instances and output to MPS or LP file.
- Set solver parameters.
- Calls LP solver for LP or MIP LP relaxation.
- Manages interaction with dynamic cut and column generators.
- Calls MIP solver.
- Returns solution and status information.
Cut Generator Library

- A collection of cutting-plane generators and management utilities.
- Interacts with OSI to inspect problem instance and solution information and get violated cuts.
- Cuts include:
  - Combinatorial cuts: AllDifferent, Clique, KnapsackCover, OddHole, ZeroHalf
  - Flow cover cuts
  - Gomory cuts: classic, GMI, RedSplit
  - Lift-and-project cuts
  - Mixed integer rounding cuts
  - General strengthening: DuplicateRows, Preprocessing, Probing, SimpleRounding
COIN LP Solver

- High-quality, efficient LP solver.
- Simplex and barrier algorithms.
- QP with barrier algorithm.
- Interface through OSI or native API.
- Tight integration with CBC (COIN-OR Branch and Cut MIP solver).
COIN Branch and Cut

- State of the art implementation of branch and cut.
- Tight integration with CLP, but can use other LP solvers through OSI.
- Uses CGL to generate cutting planes.
- Interface through OSI or native API.
- Many customization options.
The **initialize_solver()** Method

Initializing the LP solver

```cpp
#if defined(COIN_USE_CLP)

#include "OsiClpSolverInterface.hpp"
typedef OsiClpSolverInterface OsiXxxSolverInterface;

#elif defined(COIN_USE_CPX)

#include "OsiCpxSolverInterface.hpp"
typedef OsiCpxSolverInterface OsiXxxSolverInterface;

#endif

OsiSolverInterface* UFL::initialize_solver() {
    OsiXxxSolverInterface* si =
        new OsiXxxSolverInterface();
    return si;
}
```
The \texttt{create_initial_model()} Method

Creating Rim Vectors

\begin{verbatim}
CoinIotaN(integer_vars, N, M * N);
CoinFillN(col_lb, n_cols, 0.0);

int i, j, index;

for (i = 0; i < M; i++) {
    for (j = 0; j < N; j++) {
        index = xindex(i,j);
        objective[index] = trans_cost[index];
        col_ub[index] = demand[i];
    }
}
CoinFillN(col_ub + (M*N), N, 1.0);
CoinDisjointCopyN(fixed_cost, N, objective + (M * N));
\end{verbatim}
The \texttt{create\_initial\_model()} Method

Creating the Constraint Matrix

\begin{verbatim}
CoinPackedMatrix * matrix =
    new CoinPackedMatrix(false, 0, 0);

matrix->setDimensions(0, n_cols);
for (i=0; i < M; i++) { //demand constraints:
    CoinPackedVector row;
    for (j=0; j < N; j++) row.insert(xindex(i,j), 1.0);
    matrix->appendRow(row);
}

for (j=0; j < N; j++) { //linking constraints:
    CoinPackedVector row;
    row.insert(yindex(j), -1.0 * total_demand);
    for (i=0; i < M; i++) row.insert(xindex(i,j), 1.0);
    matrix->appendRow(row);
}
\end{verbatim}
Loading and Solving the LP Relaxation

Loading the Problem in the Solver

\[ \text{si->loadProblem(*matrix, col_lb, col_ub, objective, row_lb, row_ub);} \]

Solving the Initial LP Relaxation

\[ \text{si->initialSolve();} \]
The tighten_initial_model() Method

Tightening the Relaxation—Custom Cuts

```cpp
const double* sol = si->getColSolution();
int newcuts = 0, i, j, xind, yind;
for (i = 0; i < M; i++) {
    for (j = 0; j < N; j++) {
        xind = xindex(i, j);
        yind = yindex(j);

        if (sol[xind] - (demand[i] * sol[yind]) > tolerance) {
            // violated constraint
            CoinPackedVector cut;
            cut.insert(xind, 1.0);
            cut.insert(yind, -1.0 * demand[i]);
            si->addRow(cut, -1.0 * si->getInfinity(), 0.0);
            newcuts++;
        }
    }
}
```
The **tighten_initial_model()** Method

**Tightening the Relaxation—CGL Cuts**

```cpp
OsiCuts cutlist;
si->setInteger(integer_vars, N);
CglGomory * gomory = new CglGomory;
gomory->setLimit(100);
gomory->generateCuts(*si, cutlist);
CglKnapsackCover * knapsack = new CglKnapsackCover;
knapsack->generateCuts(*si, cutlist);
CglSimpleRounding * rounding = new CglSimpleRounding;
rounding->generateCuts(*si, cutlist);
CglOddHole * oddhole = new CglOddHole;
oddhole->generateCuts(*si, cutlist);
CglProbing * probe = new CglProbing;
probe->generateCuts(*si, cutlist);
si->applyCuts(cutlist);
```
Calling the Solver (Built-In MIP)

```cpp
si->setInteger(integer_vars, N);

si->branchAndBound();
if (si->isProvenOptimal()) {
    const double * solution = si->getColSolution();
    const double * objCoeff = si->getObjCoefficients();
    print_solution(solution, objCoeff, os);
}
else
    cerr << "B&B failed to find optimal" << endl;
return;
```
The **solve_model()** Method

**Calling the Solver (CLP Requires Separate MIP)**

```cpp
CbcModel model(*si);
model.branchAndBound();
if (model.isProvenOptimal()) {
    const double * solution = model.getColSolution();
    const double * objCoeff = model.getObjCoefficients();
    print_solution(solution, objCoeff, os);
}
else
    cerr << "B&B failed to find optimal" << endl;
return;
```
<project> is one of Osi, Cgl, Clp, Cbc, etc.

- **Project home pages:**
  https://projects.coin-or.org/<project> (Trac pages, soon to relocate to GitHub).

- **Documentation:**
  http://www.coin-or.org/Doxygen/<project> (Doxygen),
  http://www.coin-or.org/Clp/userguide/,
  http://www.coin-or.org/Cbc/userguide/

- **Mailing lists:**
  http://list.coin-or.org (see coin-discuss, coin-osi-devel, cgl, coin-lpsolver—note lists will be reorganized soon).