Programming Exercise / ILOG CPLEX Tutorial

Bin Hu, Günther R. Raidl, Mario Ruthmair

Algorithms and Data Structures Group
Institute of Computer Graphics and Algorithms
Vienna University of Technology

VU Algorithmics
WS 2013/14

k-Node Minimum Spanning Tree (k-MST) Problem

Given:

- (undirected) graph $G = (V, E, w)$
- nonnegative weighting function $w(e) \in \mathbb{R}_0^+, \ \forall e \in E$
- integer $k \leq |V|$

Goal: Find a minimum weight tree, spanning exactly k nodes.

Application: E.g., a cable company is allowed to serve k of n cities and wants to minimize the connection costs.

Programming Exercise: Develop integer programming formulations for this problem

Your Task (1)

- Formulate the k-**MST** problem as (mixed) integer linear programs (MILPs), based on
  1. **Miller-Tucker-Zemlin subtour elimination constraints (MTZ)**,
  2. **single commodity flows (SCF)**, and
  3. **multi commodity flows (MCF)**.
- Solve the corresponding formulations with the CPLEX solver.
- Compute the results for all 8 provided instances for $k = \lceil \frac{1}{5}|V| \rceil$ and $k = \lceil \frac{1}{2}|V| \rceil$ for all formulations.
- **For a positive grading at least 2 of 3 formulations have to achieve the correct optimal results at least for the first 4 instances!**

Your Task (2)

- Create a short **document** (preferably in LaTeX) of **about 3 pages** containing:
  1. Problem description
  2. SCF, MCF, and MTZ formulations, and descriptions of used variables and constraints
  3. Result table comparing all 3 formulations, including
     - objective function values
     - running times
     - numbers of branch-and-bound nodes
  4. Short interpretation of results
- Upload document and source code in TUWEL not later than **January 19, 2014, 23:59**.

Organization

- You should work in groups of two.
- You get an account for ADS servers (development, testing, computing results) by mail to your TISS address. Accounts will by default be deleted at end of WS2013/14! Contact us if an extension is required.
- You can work on your own computer: get CPLEX from our servers.
- We provide a **C++ framework for Linux** in TISS. You can use other programming languages in other operating systems (the CPLEX solver supports C, C++, C#, Java, and Python in Windows, Linux, and MacOS). **But we only support C++ in Linux!**

What is CPLEX?

- CPLEX: Simplex method and C programming language
- High performance (commercial) solver for
  - linear programs
  - (mixed) integer programs
  - quadratic programs
- Documentation:
  `/home1/share/ILOG/cplex-12.5/doc/html/en-US/documentation.html`
- ADS servers: `/home1/share/ILOG/cplex-12.5/`
- Home install: `/home1/share/ILOG/packages/`
- Information about ADS servers, CPLEX, etc.:
  `https://www.ads.tuwien.ac.at/w/Students`

Concert Framework

- Interface to CPLEX solver (C, C++, C#, Java, Python)
- Enables to implement: branch-and-bound, branch-and-cut, column generation, ...

Included Features:

- Preprocessing and Presolving
- Generation of general purpose and some problem-specific cuts
- Automatic branching on integer variables $\Rightarrow$ branch-and-bound

First steps in C++

1. Include the headerfile:
   `#include <ilcplex/ilocplex.h>`
2. Before the class definition (of the class that will build the model and start the solver) use the following macro:
   `ILOSTLBEGIN`
3. Declare/create the following objects:
   `IloEnv env; // the environment object`
   `IloModel model; // the model; constraints etc. go in here`
   `IloCplex cplex; // the solver object`

Basic Program Structure

```
try {
  env = IloEnv(); // create the environment
  model = IloModel(env); // create the model

  // build some variables and constraints
  // and add them to the model
  model.add(...);

  // add the objective function
  model.add(IloMinimize(...));

  cplex = IloCplex(model); // create solver object
  cplex.solve(); // solve the model
} catch (IloException& e) { ... }
catch (...) { ... }
```

Data-Types

Constants: `IloNum, IloBool, IloInt`

Variables: `IloNumVar, IloBoolVar, IloIntVar`

Example: `IloBoolVar x(env, "my-first-bool-var");`

Arrays/Vectors: `IloIntVarArray, IloNumVarArray, IloBoolVarArray`

Example:

```
// create array of 5 boolean variables
IloBoolVarArray y(env, 5);

for (u_int i=0; i<5; i++) {
  stringstream myname;
  myname << "y_" << i;
  y[i] = IloBoolVar(env, myname.str());
}
```

Constraints

Constraints, equalities, inequalities are added with help of class `IloExpr`:

Example: $y_1 + y_2 \leq 4$

```
IloExpr myExpr(env);
myExpr += y[1];
myExpr += y[2];
model.add(myExpr <= 4);
myExpr.end(); // IMPORTANT to free memory
```

Objective Function / Solver

The objective function is handled similar to constraints:

```
model.add(IloMinimize(env, expr));
```

Now, we start the LP-based branch-and-bound algorithm:

```
IloCplex cplex(model);
cplex.solve();
```

Did we succeed?

```
IloAlgorithm::Status algStatus = cplex.getStatus();
if (algStatus != IloAlgorithm::Optimal) {
  // something went wrong ...
} else {
  // model solved to optimality
  cout << "OPT: " << cplex.getObjValue() << endl;
}
// get variable values from CPLEX
IloNumArray values(env, 5);
cplex.getValues(values, y);
for (u_int i=0; i<5; i++) {
  cout << "y[" << i << "] = " << values[i] << endl;
}
```

Numeric Issues

- Due to numeric issues variable values can be within an interval of $[v - \epsilon, v + \epsilon]$ around the correct value v.
- The value of ϵ can be obtained by `cplex.getParam(IloCplex::EpInt)`.

Interpreting the CPLEX output

```
          Nodes                                                 Cuts/
Node  Left  Objective  IInf   Best Int.  Best Node  ItCnt     Gap     Variable B NodeID Parent  Depth
   0     0  12135.50     6              11258.00      2
   0     2  12246.75    19              User: 26     18                                0              0
   1     3  12311.25    17              12268.25     24                x_[96] D      1      0       1
   2     4  12338.50    17              12270.25     27                x_[51] U      2      1       2
...
* 22    22  integral     0   13210.0000 12270.25    109     7.11%      x_[82] D     22     21      18
  23    22  13083.00     6   13210.0000 12270.25    110     7.11%      x_[82] U     23     21      18
  24    22  13102.50     4   13210.0000 12270.25    112     7.11%      x_[83] U     24     23      19
* 25    21  integral     0   13109.0000 12270.25    113     6.40%      x_[86] U     25     24      20
  26    22  12372.25    19   13109.0000 12270.25    116     6.40%      x_[96] U     26      0       1
...
```

- Line 2: still in root node of B&B tree; after adding 26 inequalities the LP relaxation value is better (higher)
- Branching starts in line 3: node 1 (with parent 0) is the problem where (boolean) variable x_[96] has been set to 0 (D = down)
- First integral solution in line 5 at node 22 of B&B tree, indicated with *
- The best integer solution value is 13109, the lower bound is 12270.25.
- The gap is 6.40%; we are finished when the gap is 0% (proven optimality)

Test instances

- **Note:** The test instances include an artificial root node 0 and edges $\{0, v\}$, $\forall v \in V$, with weight 0. Why do you need this? What do you have to consider for a feasible solution?
- **We are actually interested in finding a k-MST of nodes $\{1, \ldots, |V|\}$!** Be careful, to handle this accordingly w.r.t. the number of connected nodes k!
- 8 test instances ranging from 10 to 400 nodes are included in the framework package.

Instance File Format

- First line: number of nodes (including root node)
- Second line: number of edges (including root edges)
- Subsequent lines: edge list (index, node 1, node 2, integer edge weight)

Example:

```
20
35
0  0  4  23
1  2  3  93
2  1  5  56
...
```

k-MST C++ Framework

- `Main`: starting the program, parameter handling
- `Instance`: responsible for reading the instance files, building basic data structures, and providing them to other classes
- `Tools`: provides useful functions, i.e., creating name strings for variables, measuring CPU time
- `kMST_ILP`: this class should contain the MILP-based algorithms, i.e., the formulation and the CPLEX calls

class Instance

The class Instance provides rudimentary graph data structures (which should be enough to solve the exercise):

```
struct Edge
{
  u_int v1, v2; // unordered !!!
  int weight;
};
// number of nodes and edges
u_int n_nodes, n_edges;
// array of edges
vector<Edge> edges;
// for each node we have a list of incident edges
// (represented by their indices in the array above)
vector<list<u_int> > incidentEdges;
```

Further Remarks

- First design models, then implement
- Check if final report exactly includes the used formulations
- Do not use non-linear constraints (even if it is possible in CPLEX)
- Only use variables needed in formulation, not $|V| \times |V|$ matrix
- Try to find strengthening constraints
- Use directed k-MST problem variant

Optimal Values

graph	$\|V\|$	k	OPT
g01	10	2	46
		5	477
g02	20	4	373
		10	1390
g03	50	10	725
		25	3074
g04	70	14	909
		35	3292
g05	100	20	1235
		50	4898
g06	200	40	2068
		100	6705
g07	300	60	1335
		150	4534
g08	400	80	1620
		200	5787

Table: Optimal weight values for instances g01 to g08.

Have fun coding!

Questions?

ask now or mail us