Programming Exercise / ILOG CPLEX Tutorial

Markus Leitner, Günther R. Raidl, Mario Ruthmair

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

VU Algorithmics
WS 2012

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)

- **Carefully read this tutorial**
- Formulate the k-**MST** problem as integer linear program (ILP), based on
  1. **single commodity flows (SCF)**,
  2. **multi commodity flows (MCF)**, and
  3. **Miller-Tucker-Zemlin subtour elimination constraints (MTZ)**
- Solve the corresponding formulations with ILOG CPLEX solver.

Your Task (2)

- For the implementation you should use a **C++ framework** which you can find on the course homepage.
- Furthermore you can find **8 test instances** (described later) which should be used for the evaluation and analysis of your formulations.
- Compute the results for each instance for (at least) $k = \lceil \frac{1}{5}|V| \rceil$ and $k = \lceil \frac{1}{2}|V| \rceil$ for all formulations.

Your Task (3)

- Create a short **document** (preferably in LaTeX) of **about three pages** containing:
  1. Problem description, used variables
  2. SCF, MCF and MTZ formulations
  3. Result table comparing all three formulations, including
     - objective function values
     - running times
     - numbers of branch-and-bound nodes
  4. Short interpretation of results

Organization

- You should work preferably 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 WS2012! Contact us if an extension is required.
- You can work on your own computer: get CPLEX from our servers.
- Upload document and source code in TUWEL not later than **January 20, 2013, 23:59**.
- Make sure that a correctly working version of your program is available on an ADS server at the time of your oral exam.

What is CPLEX?

- CPLEX: Simplex method and C programming language
- Commercial **optimization** software package
- High performance
  - linear programming
  - (mixed) integer programming
  - quadratic programming
- 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, ...

What does Concert do for you?

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

How can you support Concert?

- User-defined cuts
- Generate new variables (pricing) in column generation

First steps

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
```

It is important to call the `IloExpr::end()` function to free memory.

Objective Function / Solver

The objective function is handled similar to constraints:

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

Now we are ready to start the solver:

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

What is going on now?

1. Let us assume, we have defined some integer variables
2. CPLEX solves the LP relaxation of our model
3. CPLEX then tries to separate internal cutting-planes
4. If this process is finished we still may end up with a solution containing fractional variables.
5. Now it is time for *branching*; one particular variable is selected and two subproblems are created by rounding it up and down, respectively.
6. One subproblem is selected $\rightarrow$ Step 2

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 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
...
```

- Second line, still in root node of B&B tree; after the separation of 26 cuts the LP relaxation 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)

Debugging hints

- Compiling with `-O3` enables optimization. This should be used for computing final results.
- For developing use `-p -g` which includes profiling and debugging information.

Debugger:

start with `gdb --args ./yourprogram`
type `run` to start the program
type `bt` or `backtrace` to see the execution stack

This will show you e.g. the line number in which the error or segmentation fault occurs.

The tool `valgrind` can be used for the detection of memory leaks

Test instances

- The test instances already include an artificial root node 0 and edges $\{0, v\}$, $\forall v \in V$, with weight 0, needed in the flow and MTZ models. Guarantee in the model that exactly one root edge is chosen!
- **Hence, 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.
- Compute the results for each instance for (at least) $k = \lceil \frac{1}{5}|V| \rceil$ and $k = \lceil \frac{1}{2}|V| \rceil$ using all three formulations.

Instances 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 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: 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;
// incident edge indices
vector<list<u_int> > incidentEdges;
```

Graph / Algorithm libraries (optional)

- The provided (graph) methods provide all necessary functionality for this exercise.
- If you want more: code yourself or use graph / algorithm libraries like `boost` (available on the servers: `/home1/share/boost/1.49.0/` or on `http://www.boost.org`)

Further Remarks (1)

- First design models, then implement in CPLEX
- 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

Further Remarks (2)

When you work on ADS servers:

- Be nice ;-)
  when working on an ADS server start your programs with `nice` with lower priority
  ```
  $ nice ./yourprogram ...
  ```
- The 2 student servers each have 4 cores but must fit the needs of all participants, so:
  $\Rightarrow$ start only one process at a time
  $\Rightarrow$ for your final test runs make sure you get a core for your own (no more than three other processes)

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