INF580 – Large-scale Mathematical Programming

TD3 — Complexity and MP

Leo Liberti

CNRS LIX, Ecole Polytechnique, France

200124

Simple AMPL codes

Write AMPL code for the following problems:

- SUBSET-SUM
- KNAPSACK
- MAX CLIQUE
- HAMILTONIAN CYCLE

and test them with the feasible and infeasible (whenever applicable) instances given in the course slides

Random instance generators

- Coding up instances by hand is boring
- Let's use AMPL to generate random instances!
- Each problem needs its own generator
- In general, for a problem called `prob`:
  1. copy the index sets / parameters from `prob.mod` to `prob-instgen.run`
  2. set sizes by hand (e.g. "`let n := 5;`")
  3. use AMPL imperative sublanguage to randomly fill set/param values
  4. print to file

Random instance generators
Example: uniformly distributed knapsack instances

```
option randseed 0;  # pseudornd gen starts from rnd seed
param n integer, > 0;
set N := 1..n;
param c{N} integer;
param w{N} integer;
param K integer, >= 0;
## randomly generate missing index sets/params
let n := 20; # initialize number of objects
param cL := 1; param cU := 10; # bounds for object volume
param wL := 1; param wU := 10; # bounds for object value
let {i in N} c[i] := round(Uniform(cL,cU));
let {i in N} w[i] := round(Uniform(wL,wU));
let K := round((sum{i in N} c[i])/2); # generate capacity
## print out a .dat file (MIND YOU DON'T OVERWRITE OLD .dat FILES!)
print "# file generated by knapsack-instgen.run" > rndknap.dat;
printf "param n := %d;\n", n >> rndknap.dat;
printf "param K := %d;\n", K >> rndknap.dat;
printf "param : c w :=\n" >> rndknap.dat;
for {i in N} {
  printf " %i  %d %d\n", i, c[i], w[i] >> rndknap.dat;
}
printf ";\n" >> rndknap.dat;
```

Random instance generators

Example: normally distributed knapsack instances

Change

```
param cL := 1; param cU := 10; # bounds for object volume
param wL := 1; param wU := 10; # bounds for object value
let {i in N} c[i] := round(Uniform(cL,cU));
let {i in N} w[i] := round(Uniform(wL,wU));
```

to

```
param cavg := 4.5; param cstdev := 1.9; # object volume
param wavg := 5.8; param wstdev := 3.1; # object value
let {i in N} c[i] := round(Normal(cavg,cstdev));
let {i in N} w[i] := round(Normal(wavg,wstdev));
```

Random instance generators

Example: generating random graphs, Erdős-Renyi model

```
option randseed 0; # pseudornd gen starts from rnd seed
## start from sizes/index sets/params of original problem
param n integer, > 0;
set V := 1..n;
set E within {V,V};
## randomly generate missing index sets/params
let n := 50; # initialize number of vertices
param p := 0.5; # probability of creating edge
let E := {}; # initialize the edge set to empty
for {i in V, j in V : i < j} { # no loops or antiparallel arcs
  if Uniform(0,1) < p then {
    let E := E union {(i,j)}; # create the edge
  }
}
## print out a .dat file (MIND YOU DON'T OVERWRITE OLD .dat FILES!)
print "# file generated by clique-instgen.run" > rndcliq.dat;
printf "param n := %d;\n", n >> rndcliq.dat;
printf "set E :=" >> rndcliq.dat;
for {(i,j) in E} {
  printf " (%d,%d)", i,j >> rndcliq.dat;
}
printf ";\n" >> rndcliq.dat;
```

Playing with instances

- How many vertices/edges does the largest MAX CLIQUE instance have, that CPLEX can solve in 30s on your laptop?
- Generate 9 random graphs, each with 160 vertices, and with edge generation probability $p \in P = \{0.1, 0.2, \ldots, 0.9\}$
- Find max cliques on all these graphs
  - use CPLEX as a solver
  - use "`option cplex_options "mipdisplay=2";`" after "`option solver cplex;`" (shows CPLEX progress)
  - record size $\omega(G_p)$ of max clique of each graph G_p and CPU time γ_p for $p \in P$

    with bash: name random instances `rndcliq-0.1.dat`, ..., then type (1 line):

    ```
    for i in 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 ; do
      cp rndcliq-$i.dat rndcliq.dat ; ampl clique.run ;
      done > clique.log 2>&1 &
    ```

  - plot $\omega(G_p)$ versus p and γ_p versus p

    with bash: `grep OUT: clique.log | cut -d ':'  -f 2`

The Motzkin-Straus formulation

Write AMPL code to implement the Motzkin-Straus formulation for solving Max Clique

- make sure this formulation can read the same `.dat` files as those you already worked on
- test this formulation on the instance given in the course slides
- use a global optimization solver (e.g. `baron`) and also a local optimization one (e.g. `snopt`): what results do you obtain?
- can this formulation be solved using `cplex`?
- what is the maximum instance size you can solve to global optimality with this formulation? What about local optimality?

Structured formulation for SAT

- Propose a numerical encoding for SAT instances
- Based on this, write a structured MP formulation for SAT
- Implement it in AMPL and test it using an appropriate solver