Algorithmic Methods for Mathematical Models

Lab Session 3 - Mixed Integer Linear Programs

2022-2023

NIKOLAOS BELLOS

ANASTASIOS PAPAZAFEIROPOULOS

---

a)

1. These are the constants and the variables that we used

```
// Constants: Tasks, Threads, Computers, Cores
int nTasks=...;
int nThreads=...;
int nCPUs=...;
int nCores=...;
range T=1..nTasks;
range H=1..nThreads;
range C=1..nCPUs;
range K=1..nCores;
// Data: Tasks, Computers
float rh[h in H]=...;
float rc[c in C]=...;
// Data: Cores of Computer, Threads of Task
float CK[c in C, k in K]=...;
float TH[t in T, h in H]=...;
int H_t[T]; // number of threads for each task
int K_c[C]; // number of cores for each CPU
// Variables: Task in Computer, Thread in Core
dvar boolean x_tc[t in T, c in C];
dvar boolean x_hk[h in H, k in K];
dvar float+ z;
```

2. Then we calculated the |H(t)| and |K(c)| values in the following preprocessing block

```
for (var h=1; h<=nThreads; h++) {
       totalLoad += rh[h];
       for (var t=1; t<=nTasks; t++) {
              H_t[t] += TH[t][h];
       }
}

for (var c=1; c<=nCPUs; c++) {
       for (var k=1; k<=nCores; k++) {
              totalCapacity += rc[c]* CK[c][k];
              K_c[c] += CK[c][k];
       }
 }
```

3. And we applied the necessary constraints

```
// Constraint 1: Each thread should be assigned to only 1 core
forall(h in H)
  sum(k in K) x_hk[h,k] == 1;

// Constraint 2: All threads of the same task should be assigned to cores of the
same CPU
forall(t in T, c in C)
//  forall(c in C)
       sum(h in H, k in K)  x_hk[h,k]* TH[t, h]* CK[c, k] == H_t[t]* x_tc[t,c];

// Constraint 3: Capacity of one CPU should not be exceeded
forall(c in C, k in K)
       sum(h in H) rh[h]* x_hk[h,k]* CK[c,k] <= rc[c];

// Constraint 4 : Total load of all computers should not exceed the one with the
biggest load
forall(c in C)
       z >= (1/(K_c[c]* rc[c]))* sum(h in H, k in K) rh[h]* x_hk[h,k]* CK[c,k];
```

The results for the initial dataset are the ones below

```
Total load 2476.94
Total capacity 5133.39
Computers have enough capacity
Max load 53.282605185%
CPU 1 loaded at 48.393880067%
CPU 2 loaded at 41.098845828%
CPU 3 loaded at 53.282605185%
```

b)

We generated small, medium and big instances from the generator
Here are the results for the small and the medium ones
*(the free version of CPLEX does not support the big instances)

Small - 6 tasks, 36 threads, 4 CPUs, 7 cores (time: 31 ms)

```
Total load 12590.989845978
Total capacity 58482.702687028
Computers have enough capacity
Max load 22.129413749%
CPU 1 loaded at 21.098727591%
CPU 2 loaded at 21.011988093%
CPU 3 loaded at 22.129413749%
CPU 4 loaded at 20.933002441%
```

Medium - 10 tasks, 51 threads, 5 CPUs, 11 cores (time: 1.32 sec)

```
Total load 14353.080484149
Total capacity 81437.295737342
Computers have enough capacity
Max load 18.700270341%
CPU 1 loaded at 17.291424244%
CPU 2 loaded at 18.023273005%
CPU 3 loaded at 18.110978517%
CPU 4 loaded at 18.700270441%
CPU 5 loaded at 12.399820111%
```

c)

1. We have to introduce a new array variable, which will count if a processor serves any tasks or not.

```
dvar boolean y_c[c in C];
```

2. We add the required constraint to populate the array y_t[C] accordingly

```
// Constraint 4: Each CPU has flag 0 if it serves tasks and 1 elsewise
forall(c in C) {
        sum(t in T) x_tc[t,c] <= 0 => y_c[c] == 1;
        sum(t in T) x_tc[t,c] >= 1 => y_c[c] == 0;
}
```

* also we removed the constraint about the variable z since it is not required by the objective function

3. We create a new objective function to try to maximize the number of CPUs not serving any task

```
// Objective function
maximize sum(c in C) y_c[c];
```

The results for the initial dataset are the following

```
Total load 2476.94
Total capacity 5133.39
Computers have enough capacity
Total empty CPUs: 1
CPU 1 loaded at 0%
CPU 2 loaded at 48.585080474%
CPU 3 loaded at 82.779978531%
```

d) Comparison of two models P3, P3a

	P3	P3a
Solving time (initial, small, medium)	15ms, 31ms, 1.32sec	15ms, 20ms, 23ms
Number of variables	nTasks*nCPUs + nThreads*nCores	nTasks*nCPUs + nThreads*nCores + nCPUs
Number of constraints	4	4