Adding DCAP to `Siplib.jl`

Yongkyu Cho

POSTECH
E-mail: jyg1124@postech.ac.kr

June 25, 2018

1 Assumptions

- You have `Julia` $\geq$ 0.6.2. You know some basic `Julia` syntax.
- You know how to model with JuMP.
- You have a two-stage stochastic programming problem (e.g., DCAP A) that you want to add to `Siplib.jl`. You know everything about the problem, e.g., formulation, probability distributions.
- Your `Siplib.jl` package is located in `/dir/Siplib`.

2 Tutorial

Follow the steps to add a new problem. Be very careful whenever you make a new component that is named based on the problem name (e.g., folder, script, function, composite-type). You must be consistent with the predefined manner. Otherwise, it will cause error.

2.1 Modify and save the file: `/dir/Siplib/src/problem_info.csv`

`problem_info.csv` file contains parameter information for each problem. Open the file and add the following comma-delimited line at the bottom:

```
DCAP,4,"[R, N, T, S], All integers."
```

- `DCAP`: Name of the problem.
- `4`: Number of parameters for the problem.
- `"[R, N, T, S], All integers."`: You can put any note on the parameters. If you do not feel like to put anything, just let it `""`.

Later, SMPS files will be generated using the function with at least two input arguments, e.g.,:

```
julia> generateSMPS(:DCAP,[3,3,3,10])
```

Note that the function `generateSMPS()` takes `Symbol`-type argument `:DCAP` and `Array`-type argument `[3,3,3,10]`.

2.2 Create a new folder DCAP under: /dir/Siplib/src/problem/

So, you must have a directory: `/dir/Siplib/src/problem/DCAP/`

2.3 Create, Write, and Place DCAP_data.jl file into: /dir/Siplib/src/problem/DCAP/

Make a new julia script file using the problem name. For DCAP, it must be `DCAP_data.jl`. This file must contain two definitions with predefined naming rule and set of input arguments. For `DCAP_data.jl`,

- Definition of a composite-type `DCAPData` that will contain instance data.
- Definition of a function

  `DCAPData(nR::Int, nN::Int, nT::Int, nS::Int, seed::Int)::DCAPData`

  that will generate data and return it with a `DCAPData`-type object. The argument `seed` must be placed at the end. Later, the function `generateSMPS()` will call this function taking two arguments `:DCAP` and `[R,N,T,S]` to get an instance. Therefore, the order of the parameters must always be consistent.

For example, `DCAP_data.jl` must be written by (please see the comment for each line):

```
# definition of data container
mutable struct DCAPData # 'mutable struct' is Julia's composite type
    # Sets
    R # set of resources (i ∈ R)
    N # set of tasks (j ∈ N)
    T # set of time periods (t ∈ T)
    S # set of scenarios (s ∈ S)

    # Parameters
    a # a[i,t]: variable cost for expanding capacity of resource i at time t
    b # b[i,t]: fixed cost for expanding capacity of resource i at time t
    c # c[i,j,t,s]: cost of processing task j using resource i in period t under scenario s
    c0 # c0[j,t,s]: penalty cost of failing to assign a resource to task j under scenario s
    d # d[j,t,s]: processing requirement for task j in period t under scenario s

    Pr # Pr[s]: probability of occurence of scenario s

    DCAPData() = new() # constructor
end

# data generating function
## nR: number of resources, nN: number of tasks, nT: number of times, nS: number of scenarios, seed: random seed
function DCAPData(nR::Int, nN::Int, nT::Int, nS::Int, seed::Int)::DCAPData

    srand(seed) # set a random seed
    data = DCAPData() # construct empty container

    # store Sets
    data.R = 1:nR # set range for R
    data.N = 1:nN # set range for N
    data.T = 1:nT # set range for T
    data.S = 1:nS # set range for S
    # store Parameters
    data.a = rand(nR, nT) * 5 + 5 # generate and store random data
    data.b = rand(nR, nT) * 40 + 10 # generate and store random data
    data.c = rand(nR, nN, nT, nS) * 5 + 5 # generate and store random data
    data.c0 = rand(nN, nT, nS) * 500 + 500 # generate and store random data
    data.d = rand(nN, nT, nS) + 0.5 # generate and store random data

    data.Pr = ones(nS)/nS # store probability of occurence of scenarios

    return data
end
```

2.4 Create, Write, and Place `DCAP_model.jl` file into: `/dir/Siplib/src/problem/DCAP/`

Make a new julia script file using the problem name again. For DCAP, it must be `DCAP_model.jl`. This file must include the file `DCAP_data.jl` we created above. This file must also contain a definition of a function with the name `DCAP`.

- Inclusion of the `Julia` script: `DCAP_data.jl`.
- Definition of a function

  `DCAP(nR::Int, nN::Int, nT::Int, nS::Int, seed::Int=1)::JuMP.Model`

  that will return a `JuMP.Model`-type object. The set of arguments and its order must be the same as `DCAPData()` except for the `seed` that must have a default value 1. Don't forget to set `seed::Int=1`. In this function, `seed` is an optional argument.

For example, `DCAP_data.jl` must be written as the following script. Please see the comment for each line to learn how to model a stochastic program with StructJuMP syntax.

```
include("./DCAP_data.jl") # Do not forget to include DCAP_data.jl file we defined above

# JuMP.Model constructing function
function DCAP(nR::Int, nN::Int, nT::Int, nS::Int, seed::Int=1)::JuMP.Model

    # generate instance data
    data = DCAPData(nR, nN, nT, nS, seed)

    # copy for readability
    R, N, T, S = data.R, data.N, data.T, data.S
    a, b, c, c0, d, Pr = data.a, data.b, data.c, data.c0, data.d, data.Pr

    # construct JuMP.Model using StructuredModel() defined in StructJuMP package
    model = StructuredModel(num_scenarios = nS) # the keyword argument: num_scenarios = nS

    ## Set 1st stage components using JuMP syntax
    @variable(model, x[i=R,t=T] >= 0)
    @variable(model, u[i=R,t=T], Bin)
    @objective(model, Min, sum(a[i,t]*x[i,t] + b[i,t]*u[i,t] for i in R for t in T))
    @constraint(model, [i=R,t=T], x[i,t] - u[i,t] <= 0)

    ## Set 2st stage components using StructJuMP syntax
    for s in S # for all the scenarios
        # declare a StructuredModel 'sb' (sibling) that must have three keyword arguments: parent=model (defined above), id
            =s, prob=Pr[s]
        sb = StructuredModel(parent=model, id=s, prob=Pr[s])
        # declare 2nd stage variables. You must not put the scenario−specific index for the 2nd stage variables. They are
            internally distiguished by 'id' (one of the keyword arguments of StructuredModel()).
        @variable(sb, y[i=R, j=N, t=T], Bin)
        @variable(sb, z[j=N,t=T] >= 0)
        # declare the recourse objective function using JuMP syntax. Again, you don't need to worry about taking expectation
            . But, be very careful about indices of the scenario data containers (i.e., c[i,j,t,s] and c0[j,t,s]). They need to be
            explicit since they are not members of StructuredModel.
        @objective(sb, Min,
              sum(c[i,j,t,s]*y[i,j,t] for i in R for j in N for t in T) +
              sum(c0[j,t,s]*z[j,t] for j in N for t in T)
        )
        # declare the 2nd stage constrains using JuMP syntax. Again, be very careful about indices of the scenario data
             container (i.e., d[j,t,s])
        @constraint(sb, [i=R, t=T],
                -sum(x[i,tau] for tau in 1:t) + sum(d[j,t,s]*y[i,j,t] for j in N) <= 0
        )
        @constraint(sb, [j=N, t=T],
                sum(y[i,j,t] for i in R) + z[j,t] == 1
        )
    end

    return model
end
```

2.5 Check if everything is well done

Your directory `/dir/Siplib/src/problem/DCAP/` must contain the two `Julia` scripts.

```
Siplib
└ src
   ├ problems
   │  └ DCAP
   │     ├─ DCAP_data.jl
   │     └─ DCAP_models.jl
   ├─ Siplib.jl
   ├─ smpswriter.jl
   ├─ generator.jl
   ├─ analyzer.jl
   └─ utility.jl
```

Then, try the following steps.

1. Open a terminal and change working directory to `dir/Siplib/src/`:

```
user@LINUX:~$ cd dir/Siplib/src
```

2. Run `Julia` in that directory. For example,

```
user@LINUX:~/dir/Siplib/src$ julia
```

3. Excute `include("Siplib.jl")`

```
julia> include("Siplib.jl")
```

4. Excute `using Siplib`

```
julia> using Siplib
```

5. Execute the following line to generate DCAP_2_2_2_10 instance (don't forget to put colon in front of the problem name):

```
julia> generateSMPS(:DCAP, [2,2,2,10])
```

If it works well, you will see the three SMPS files in `dir/Siplib/instance`:

```
Siplib
└ instance
   ├─DCAP_2_2_2_10.cor
   ├─DCAP_2_2_2_10.tim
   └─DCAP_2_2_2_10.sto
```

Appendix

A DCAP: Dynamic capacity planning with stochastic demand

DCAP is the problem of determining a capacity expansion schedule for a set of resources, and the assignment of resource capacity to task with stochastic requirement over a multi-period planning horizon.

A.1 DCAP: Mathematical formulation

We consider the problem of deciding the capacity expansion schedule for $|R|$ resources over $|T|$ time periods to satisfy the processing requirements of $|N|$ tasks where R, T, and N denote set of resources, set of time periods, and set of tasks, respectively. We define decision variables: the first-stage continuous variable x_{it} for the capacity acquisition of resource i in period t and the second-stage binary variable y^s_{ijt} to indicate whether resource i is assigned to task j in period t under scenario s. Additional first-stage binary variable u_{it} is for logical constraint whether or not we decided to acquire more capacity of resource i in period t. Hence, for all resource $i \in R$ and time $t \in T$, $u_{it} = 1$ if $x_{it} > 0$, $u_{it} = 0$ otherwise.

Under the definition of the decision variables, the extensive form of DCAP is written below and the summarized notation is available in Table 1.

$$\text{(DCAP)} \min \sum_{t\in T}\sum_{i\in R}(\alpha_{it}x_{it}+\beta_{it}u_{it})+\sum_{s\in\mathcal{S}}\mathbb{P}(s)\sum_{t\in T}\sum_{i\in R\cup\{0\}}\sum_{j\in N}c^s_{ijt}y^s_{ijt} \tag{1a}$$

$$\text{s.t. } x_{it}\le \mathrm{M}u_{it}, \quad \forall i\in R,\ \forall t\in T, \tag{1b}$$

$$\sum_{j\in N} d^s_{jt}y^s_{ijt}\le\sum_{\tau=1}^{t}x_{i\tau}, \quad \forall i\in R,\ \forall t\in T,\ \forall s\in\mathcal{S}, \tag{1c}$$

$$\sum_{i\in R\cup\{0\}} y^s_{ijt}=1, \quad \forall j\in N,\ \forall t\in T,\ \forall s\in\mathcal{S}, \tag{1d}$$

$$x_{it}\ge 0, \quad \forall i\in R,\ \forall t\in T, \tag{1e}$$

$$u_{it}\in\{0,1\}, \quad \forall i\in R,\ \forall t\in T, \tag{1f}$$

$$y^s_{ijt}\in\{0,1\}, \quad \forall i\in R\cup\{0\},\ \forall j\in N,\ \forall t\in T,\ \forall s\in\mathcal{S}, \tag{1g}$$

The objective function (1a) is to minimize total expected cost for the capacity expansion schedule. The first double summation denotes the expansion cost for resource i in period t where α_{it} and β_{it} are the variable and fixed cost, respectively. The second term in the objective function represents the expected assignment cost in period t over all scenario $s\in\mathcal{S}$. Note that a dummy resource $i=0$ is included with infinite capacity. The cost c^s_{0jt} denotes the penalty of failing to assign a resource to task j. The dummy resource enforces the *complete recourse property*, which ensures that there is a feasible second-stage assignment in all periods and all scenarios for any capacity acquisition schedule. Constraint (1b) is the logical constraint containing a suitably large value M (we set M=1 in SIPLIB 2.0 to follow the original implementation in SIPLIB although it does not seem to be large enough) to define the cost for capacity expansion. Constraint (1c) reflects that the processing requirement of all tasks assigned to a resource in any period cannot exceed the installed capacity in that period under all scenarios. Constraint (1d) guarantees that each task needs to be assigned to exactly one resource in each period under all scenarios. Finally, constraints (1e)-(1g) restrict the space from which the variables take values.

Table 1: Notations for DCAP

Index sets:	
R	index set of resources ($i \in R \cup \{0\}$ where 0 is a dummy resource with infinite capacity)
N	index set of tasks ($j \in N$)
T	index set of time periods ($t \in T$)
$\mathcal{S}$	index set of scenarios ($s \in \mathcal{S}$)
Parameters:	
α_{it}	variable cost for expanding capacity of resource i
β_{it}	fixed cost for expanding capacity of resource i
c^s_{ijt}	cost of processing task j using resource i in period t under scenario s
d^s_{jt}	processing requirement for task j in period t under scenario s
$\mathbb{P}(s)$	the probability of occurence of scenario s
Decision variables:	
x_{it} (1st-stage)	capacity acquisition amount of resource i in period t
u_{it} (1st-stage)	1 if capacity of resource i is expanded in period t, 0 otherwise
y^s_{ijt} (2nd-stage)	1 if resource i is assigned to task j in period t under scenario s, 0 otherwise

A.2 DCAP: Data generation

There are four factors that define the instance of DCAP: $|R|$, $|N|$, $|T|$, and $|\mathcal{S}|$. Once we decide the factors, the instance is named by DCAP_$|R|$_$|N|$_$|T|$_$|\mathcal{S}|$. Let U be a continuous uniform random variable: $U \sim Unif(0,1)$. Then, the parameters are generated as follows:

$$\begin{aligned}
\alpha_{it} &= 5U + 5, \quad \forall i \in R,\ \forall t \in T, \\
\beta_{it} &= 40U + 10, \quad \forall i \in R,\ \forall t \in T, \\
c^s_{ijt} &= 5U + 5, \quad \forall i \in R,\ \forall j \in N,\ \forall t \in T,\ \forall s \in \mathcal{S}, \\
c^s_{0jt} &= 500U + 500, \quad \forall j \in N,\ \forall t \in T,\ \forall s \in \mathcal{S}, \\
d^s_{jt} &= U + 0.5, \quad \forall j \in N,\ \forall t \in T,\ \forall s \in \mathcal{S}.
\end{aligned}$$