Multistage investment planning for renewable electricity systems

Andy Philpott

Joint work with Anthony Downward and Regan Baucke

Electric Power Optimization Centre
University of Auckland
www.epoc.org.nz

International Conference on Stochastic Programming, Davis, July 27, 2023.

Outline

Transitioning to renewable electricity

Multi-horizon modelling framework

- Multi-horizon stochastic programming and capacity planning
- The JuDGE package

The EMERALD Model

- New Zealand case study
- Results

Outline

Transitioning to renewable electricity

Multi-horizon modelling framework

- Multi-horizon stochastic programming and capacity planning
- The JuDGE package

The EMERALD Model

- New Zealand case study
- Results

Transitioning to renewable electricity

Many models have been developed to plan for future zero-carbon energy systems.

- Deterministic and stochastic planning models;
- Agent simulation models;

[2] Ralph & Smeers, SIOPT, 2015, Ferris & P., Operations Research, 2022

Transitioning to renewable electricity

Many models have been developed to plan for future zero-carbon energy systems.

- Deterministic and stochastic planning models;
- Agent simulation models;
- Competitive equilibrium ?

[2] Ralph & Smeers, SIOPT, 2015, Ferris & P., Operations Research, 2022

Transitioning to renewable electricity

Many models have been developed to plan for future zero-carbon energy systems.

- Deterministic and stochastic planning models;
- Agent simulation models;
- Competitive equilibrium ?

Need to model risk: generation capacity choices are made by risk-averse commercial investors responding to incentives (carbon prices) and/or regulation.

Optimal risk-averse plan matches partial equilibrium when risk measures are coherent and risk-trading instruments available.[2]

[2] Ralph & Smeers, SIOPT, 2015, Ferris & P., Operations Research, 2022

Outline

Transitioning to renewable electricity

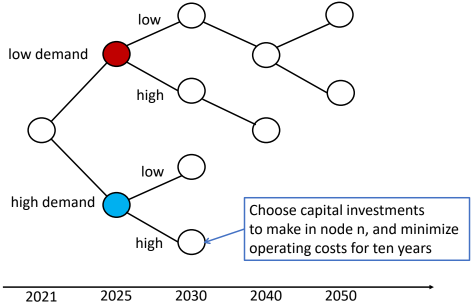
Multi-horizon modelling framework

- Multi-horizon stochastic programming and capacity planning
- The JuDGE package

The EMERALD Model

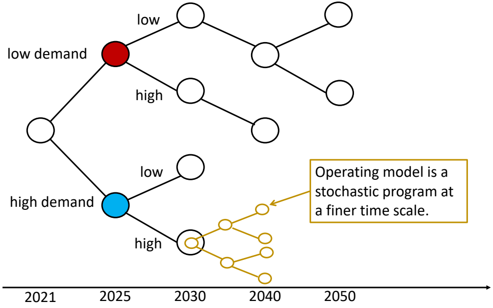
- New Zealand case study
- Results

Multi-horizon planning



Capacity-expansion decisions over longer time scale (5 years or 10 years) result in lower operational costs, or higher revenue in the future.

Multi-horizon scenario trees [Kaut et al, 2014]



Operational decisions with short-term uncertainty optimized by a stochastic program.

General formulation

- $\mathcal{N}$ is the set of nodes in the scenario tree;
- ϕ_n the probability of the state of the world n occurring;
- $\mathcal{P}_n$ the set of nodes on the path to (and including) node n;
- m is the number of expansion variables;
- $z_n \in \mathcal{Z}_+^m$ are the variables for the expansions made at node n;
- y_n is the variable vector for stage-problem n;
- $\mathcal{Y}_n$ is the stage-problem feasibility set.

Extensive Form:

$$\begin{aligned}
\min_{y,z} \quad & \sum_{n \in \mathcal{N}} \phi_n (c_n^\top z_n + q_n^\top y_n) \\
\text{s.t.} \quad & A_n y_n \leq b + D \sum_{h \in \mathcal{P}_n} z_h, && \forall n \in \mathcal{N}, \\
& y_n \in \mathcal{Y}_n, && \forall n \in \mathcal{N}, \\
& z_n \in \mathcal{Z}_+^m, && \forall n \in \mathcal{N}.
\end{aligned}$$

General formulation

- $\mathcal{N}$ is the set of nodes in the scenario tree;
- ϕ_n the probability of the state of the world n occurring;
- $\mathcal{P}_n$ the set of nodes on the path to (and including) node n;
- m is the number of expansion variables;
- $z_n \in \mathcal{Z}_+^m$ are the variables for the expansions made at node n;
- y_n is the variable vector for stage-problem n;
- $\mathcal{Y}_n$ is the stage-problem feasibility set.

Extensive Form:

$$
\begin{aligned}
\min_{y,z} \quad & \sum_{n \in \mathcal{N}} \phi_n (c_n^\top z_n + q_n^\top y_n) \\
\text{s.t.} \quad & A_n y_n \leq b + D \sum_{h \in \mathcal{P}_n} z_h, && \forall n \in \mathcal{N}, \\
& y_n \in \mathcal{Y}_n, && \forall n \in \mathcal{N}, \\
& z_n \in \mathcal{Z}_+^m, && \forall n \in \mathcal{N}.
\end{aligned}
$$

General formulation

- $\mathcal{N}$ is the set of nodes in the scenario tree;
- ϕ_n the probability of the state of the world n occurring;
- $\mathcal{P}_n$ the set of nodes on the path to (and including) node n;
- m is the number of expansion variables;
- $z_n \in \mathcal{Z}_+^m$ are the variables for the expansions made at node n;
- y_n is the variable vector for stage-problem n;
- $\mathcal{Y}_n$ is the stage-problem feasibility set.

Extensive Form:

$$
\begin{aligned}
\min_{y,z} \quad & \sum_{n \in \mathcal{N}} \phi_n (c_n^\top z_n + q_n^\top y_n) \\
\text{s.t.} \quad & A_n y_n \leq b + D \sum_{h \in \mathcal{P}_n} z_h, && \forall n \in \mathcal{N}, \\
& y_n \in \mathcal{Y}_n, && \forall n \in \mathcal{N}, \\
& z_n \in \mathcal{Z}_+^m, && \forall n \in \mathcal{N}.
\end{aligned}
$$

General formulation

- $\mathcal{N}$ is the set of nodes in the scenario tree;
- ϕ_n the probability of the state of the world n occurring;
- $\mathcal{P}_n$ the set of nodes on the path to (and including) node n;
- m is the number of expansion variables;
- $z_n \in \mathcal{Z}_+^m$ are the variables for the expansions made at node n;
- y_n is the variable vector for stage-problem n;
- $\mathcal{Y}_n$ is the stage-problem feasibility set.

Extensive Form:

$$
\begin{aligned}
\min_{y,z} \quad & \sum_{n\in\mathcal{N}} \phi_n(c_n^\top z_n + q_n^\top y_n) \\
\text{s.t.} \quad & A_n y_n \le b + D \sum_{h\in\mathcal{P}_n} z_h, && \forall n \in \mathcal{N}, \\
& y_n \in \mathcal{Y}_n, && \forall n \in \mathcal{N}, \\
& z_n \in \mathcal{Z}_+^m, && \forall n \in \mathcal{N}.
\end{aligned}
$$

General formulation

- $\mathcal{N}$ is the set of nodes in the scenario tree;
- ϕ_n the probability of the state of the world n occurring;
- $\mathcal{P}_n$ the set of nodes on the path to (and including) node n;
- m is the number of expansion variables;
- $z_n \in \mathcal{Z}_+^m$ are the variables for the expansions made at node n;
- y_n is the variable vector for stage-problem n;
- $\mathcal{Y}_n$ is the stage-problem feasibility set.

Extensive Form:

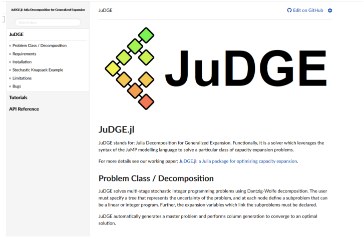
$$\begin{aligned}
\min_{y,z} \quad & \sum_{n \in \mathcal{N}} \phi_n (c_n^\top z_n + q_n^\top y_n) \\
\text{s.t.} \quad & A_n y_n \leq b + D \sum_{h \in \mathcal{P}_n} z_h, && \forall n \in \mathcal{N}, \\
& y_n \in \mathcal{Y}_n, && \forall n \in \mathcal{N}, \\
& z_n \in \mathcal{Z}_+^m, && \forall n \in \mathcal{N}.
\end{aligned}$$

General formulation

- $\mathcal{N}$ is the set of nodes in the scenario tree;
- ϕ_n the probability of the state of the world n occurring;
- $\mathcal{P}_n$ the set of nodes on the path to (and including) node n;
- m is the number of expansion variables;
- $z_n \in \mathcal{Z}_+^m$ are the variables for the expansions made at node n;
- y_n is the variable vector for stage-problem n;
- $\mathcal{Y}_n$ is the stage-problem feasibility set.

Extensive Form:

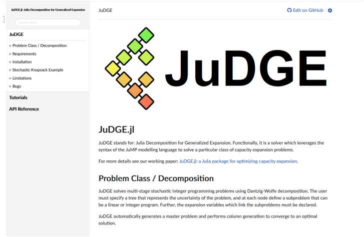
$$
\begin{aligned}
\min_{y,z} \quad & \sum_{n \in \mathcal{N}} \phi_n (c_n^\top z_n + q_n^\top y_n) \\
\text{s.t.} \quad & A_n y_n \leq b + D \sum_{h \in \mathcal{P}_n} z_h, && \forall n \in \mathcal{N}, \\
& y_n \in \mathcal{Y}_n, && \forall n \in \mathcal{N}, \\
& z_n \in \mathcal{Z}_+^m, && \forall n \in \mathcal{N}.
\end{aligned}
$$



`https://github.com/EPOC-NZ/JuDGE.jl`

JuDGE stands for Julia Decomposition for Generalized Expansion.).

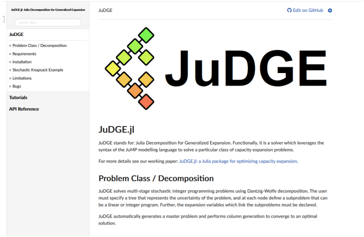
- allows users to easily implement multi-horizon optimization models using the JuMP modelling language;
- can apply end-of-horizon risk-measures in objective function and/or the constraints; and
- outputs an interactive view of the results over the scenario tree, enabling decision makers to explore the optimal expansion plan.



`https://github.com/EPOC-NZ/JuDGE.jl`

JuDGE stands for Julia Decomposition for Generalized Expansion.).

- allows users to easily implement multi-horizon optimization models using the JuMP modelling language;
- can apply end-of-horizon risk-measures in objective function and/or the constraints; and
- outputs an interactive view of the results over the scenario tree, enabling decision makers to explore the optimal expansion plan.



`https://github.com/EPOC-NZ/JuDGE.jl`

JuDGE stands for Julia Decomposition for Generalized Expansion.).

- allows users to easily implement multi-horizon optimization models using the JuMP modelling language;
- can apply end-of-horizon risk-measures in objective function and/or the constraints; and
- outputs an interactive view of the results over the scenario tree, enabling decision makers to explore the optimal expansion plan.

JuDGE modelling framework

To apply JuDGE we require...

- a tree with corresponding data and probabilities for each node;
- a subproblem defined as a JuMP model for each node in the tree; and
- expansion (and/or shutdown) decisions and costs;
- a choice of solver for master and subproblem.

JuDGE automatically forms a restricted master problem, and applies Dantzig-Wolfe decomposition.[3]

The LP relaxation of the restricted master problem is typically solved with an interior point method, and the subproblems are solved as mixed-integer programs.

JuDGE can formulate the deterministic equivalent problem directly as a JuMP model (mixed-integer program).

[3] Singh, P. & Wood, *Operations Research, 2009.*

JuDGE modelling framework

To apply JuDGE we require...

- a tree with corresponding data and probabilities for each node;
- a subproblem defined as a JuMP model for each node in the tree; and
- expansion (and/or shutdown) decisions and costs;
- a choice of solver for master and subproblem.

JuDGE automatically forms a restricted master problem, and applies Dantzig-Wolfe decomposition.[3]

The LP relaxation of the restricted master problem is typically solved with an interior point method, and the subproblems are solved as mixed-integer programs.

JuDGE can formulate the deterministic equivalent problem directly as a JuMP model (mixed-integer program).

[3] Singh, P. & Wood, *Operations Research, 2009.*

JuDGE modelling framework

To apply JuDGE we require...

- a tree with corresponding data and probabilities for each node;
- a subproblem defined as a JuMP model for each node in the tree; and
- expansion (and/or shutdown) decisions and costs;
- a choice of solver for master and subproblem.

JuDGE automatically forms a restricted master problem, and applies Dantzig-Wolfe decomposition.[3]

The LP relaxation of the restricted master problem is typically solved with an interior point method, and the subproblems are solved as mixed-integer programs.

JuDGE can formulate the deterministic equivalent problem directly as a JuMP model (mixed-integer program).

[3]Singh, P. & Wood, *Operations Research, 2009.*

JuDGE modelling framework

To apply JuDGE we require...

- a tree with corresponding data and probabilities for each node;
- a subproblem defined as a JuMP model for each node in the tree; and
- expansion (and/or shutdown) decisions and costs;
- a choice of solver for master and subproblem.

JuDGE automatically forms a restricted master problem, and applies Dantzig-Wolfe decomposition.[3]

The LP relaxation of the restricted master problem is typically solved with an interior point method, and the subproblems are solved as mixed-integer programs.

JuDGE can formulate the deterministic equivalent problem directly as a JuMP model (mixed-integer program).

[3]Singh, P. & Wood, *Operations Research, 2009.*

JuDGE modelling framework

To apply JuDGE we require...

- a tree with corresponding data and probabilities for each node;
- a subproblem defined as a JuMP model for each node in the tree; and
- expansion (and/or shutdown) decisions and costs;
- a choice of solver for master and subproblem.

JuDGE automatically forms a restricted master problem, and applies Dantzig-Wolfe decomposition.[3]

The LP relaxation of the restricted master problem is typically solved with an interior point method, and the subproblems are solved as mixed-integer programs.

JuDGE can formulate the deterministic equivalent problem directly as a JuMP model (mixed-integer program).

[3] Singh, P. & Wood, *Operations Research, 2009.*

JuDGE modelling framework

To apply JuDGE we require...

- a tree with corresponding data and probabilities for each node;
- a subproblem defined as a JuMP model for each node in the tree; and
- expansion (and/or shutdown) decisions and costs;
- a choice of solver for master and subproblem.

JuDGE automatically forms a restricted master problem, and applies Dantzig-Wolfe decomposition.[3]

The LP relaxation of the restricted master problem is typically solved with an interior point method, and the subproblems are solved as mixed-integer programs.

JuDGE can formulate the deterministic equivalent problem directly as a JuMP model (mixed-integer program).

[3] Singh, P. & Wood, *Operations Research, 2009.*

JuDGE modelling framework

To apply JuDGE we require...

- a tree with corresponding data and probabilities for each node;
- a subproblem defined as a JuMP model for each node in the tree; and
- expansion (and/or shutdown) decisions and costs;
- a choice of solver for master and subproblem.

JuDGE automatically forms a restricted master problem, and applies Dantzig-Wolfe decomposition.[3]

The LP relaxation of the restricted master problem is typically solved with an interior point method, and the subproblems are solved as mixed-integer programs.

JuDGE can formulate the deterministic equivalent problem directly as a JuMP model (mixed-integer program).

[3] Singh, P. & Wood, *Operations Research, 2009.*

Outline

Transitioning to renewable electricity

Multi-horizon modelling framework

- Multi-horizon stochastic programming and capacity planning
- The JuDGE package

The EMERALD Model

- New Zealand case study
- Results

EMERALD: New Zealand case study demo

The EMERALD model is designed to study effects of carbon-prices and explicit restrictions on non-renewables.

Case study uses...

- Three regions.
- Four seasons with 10 load blocks each.
- 16 load growth scenarios.
- 13 historical years model seasonal hydrological inflows.
- Data based on two-stage model of NZ system.[4]
- Assume risk neutrality for simplicity.

[4] Ferris & Philpott, 100% renewable electricity with storage (2019) http://www.epoc.org.nz.

New Zealand case study

EMERALD input data

Scenario tree for demand

```
mytree, data = tree_with_data(myscenariotree.csv)
```

```
n,p,EVTWh,industryTWh,consumerTWh,TiwauTWh,carbon
1,-,0.1,8.525,27.727,5.475,50
11,1,0.1389,10.750025,32.16332,5.475,50
12,1,0.1389,11.50875,29.168804,5.475,50
111,11,0.55,12.276,35.49056,5.475,50
112,11,0.55,11.227425,28.55881,5.475,50
121,12,0.55,13.14555,32.191047,5.475,50
122,12,0.55,12.028775,25.897018,5.475,50
1111,111,5,15.8565,39.566429,5.475,50
1121,112,5,14.50955,31.802869,5.475,50
....
```

EMERALD input data

Scenario tree for demand

```
mytree, data = tree_with_data(myscenariotree.csv)

 n,p,EVTWh,industryTWh,consumerTWh,TiwauTWh,carbon
 1,-,0.1,8.525,27.727,5.475,50
 11,1,0.1389,10.750025,32.16332,5.475,50
 12,1,0.1389,11.50875,29.168804,5.475,50
 111,11,0.55,12.276,35.49056,5.475,50
 112,11,0.55,11.227425,28.55881,5.475,50
 121,12,0.55,13.14555,32.191047,5.475,50
 122,12,0.55,12.028775,25.897018,5.475,50
 1111,111,5,15.8565,39.566429,5.475,50
 1121,112,5,14.50955,31.802869,5.475,50
 ....

JuDGE.visualize_tree(mytree, data)
```

EMERALD input data

Scenario tree for demand

```
mytree, data = tree_with_data(myscenariotree.csv)

 n,p,EVTWh,industryTWh,consumerTWh,TiwauTWh,carbon
 1,-,0.1,8.525,27.727,5.475,50
 11,1,0.1389,10.750025,32.16332,5.475,50
 12,1,0.1389,11.50875,29.168804,5.475,50
 111,11,0.55,12.276,35.49056,5.475,50
 112,11,0.55,11.227425,28.55881,5.475,50
 121,12,0.55,13.14555,32.191047,5.475,50
 122,12,0.55,12.028775,25.897018,5.475,50
 1111,111,5,15.8565,39.566429,5.475,50
 1121,112,5,14.50955,31.802869,5.475,50
 ....

JuDGE.visualize_tree(mytree, data)
```

EMERALD input data

Setting the solvers

```
env = Gurobi.Env()

JuDGE_SP_Solver = optimizer_with_attributes(() ->
Gurobi.Optimizer(env), "OutputFlag" => 0, "MIPGap" =>
0.0)
```

EMERALD input data

Setting the solvers

```
env = Gurobi.Env()

JuDGE_SP_Solver = optimizer_with_attributes(() ->
Gurobi.Optimizer(env), "OutputFlag" => 0, "MIPGap" =>
0.0)

 JuDGE_MP_Solver = optimizer_with_attributes(() ->
Gurobi.Optimizer(env), "OutputFlag" => 0, "Method" =>
2, "Crossover" => 0, "MIPGap" => 0.0)
```

EMERALD input data

Setting the solvers

```
env = Gurobi.Env()

JuDGE_SP_Solver = optimizer_with_attributes(() ->
Gurobi.Optimizer(env), "OutputFlag" => 0, "MIPGap" =>
0.0)

 JuDGE_MP_Solver = optimizer_with_attributes(() ->
Gurobi.Optimizer(env), "OutputFlag" => 0, "Method" =>
2, "Crossover" => 0, "MIPGap" => 0.0)
```

EMERALD input data

Defining subproblems

```
function sub_problems(n::AbstractTree)

sp = JuMP.Model(JuDGE_SP_Solver)

@expansion(sp, 0 <= investment[k in 1..invest_keys] <=
numUnits[k], lag = 1)#, Int)

@capitalcosts(sp, lifetime[n] *
sum(sum(l.generators[g].capitalcosts *
l.generators[g].investment
* investment[(i, g)] / l.generators[g].numUnits
for g in keys(l.generators)) for (i, l) in
data[n].locations))
```

EMERALD input data

Defining subproblems

```
function sub_problems(n::AbstractTree)

sp = JuMP.Model(JuDGE_SP_Solver)

@expansion(sp, 0 <= investment[k in 1..invest_keys] <=
numUnits[k], lag = 1)#, Int)

@capitalcosts(sp, lifetime[n] *
sum(sum(l.generators[g].capitalcosts *
l.generators[g].investment
* investment[(i, g)] / l.generators[g].numUnits
for g in keys(l.generators)) for (i, l) in
data[n].locations))
```

EMERALD input data

Defining subproblems

```
@variable(sp, output[gen_keys] >= 0)
@variable(sp, CO2emission[gen_keys] >= 0)
@variable(sp, 0 <= flow[(l, k, j, b, h) in flow_keys]
<= data[n].network[(l, k)])
@variable(sp, target[(l, t) in target_keys] >= 0)
@variable(sp, storage[(l, t, h) in storage_keys] >= 0)
.......
```

EMERALD input data

Defining subproblems

```
@objective(sp, Min, sum(10000 * investment[(l, g)]
for (l, g) in invest_keys) +
sum(data[n].locations[l].generators[g].SRMC *
output[(l, t, b, h, g)]
* ...  for (l, t, b, h, g) in gen_keys) ))
```

EMERALD input data

Defining subproblems

```
@objective(sp, Min, sum(10000 * investment[(l, g)]
for (l, g) in invest_keys) +
sum(data[n].locations[l].generators[g].SRMC *
output[(l, t, b, h, g)]
* ...  for (l, t, b, h, g) in gen_keys) ))

@constraint(sp,load_balance[(l, t, b, h) in
shed_keys],
data[n].locations[l].loadblocks[b].load *
data[n].seasons[t].demand - shedding[(l, t, b, h)] ==
sum(output[(l, t, b, h, g)] for g in
keys(data[n].locations[l].generators)) ...
```

EMERALD input data

Creating the JuDGE model

```
model = JuDGEModel(mytree,
          ConditionallyUniformProbabilities,
          sub_problems,
          JuDGE_MP_Solver,
          discount_factor=0.92)
```

EMERALD input data

Creating the JuDGE model

```
model = JuDGEModel(mytree,
          ConditionallyUniformProbabilities,
          sub_problems,
          JuDGE_MP_Solver,
          discount_factor=0.92)
```

EMERALD input data

Creating the JuDGE model

```
model = JuDGEModel(mytree,
          ConditionallyUniformProbabilities,
          sub_problems,
          JuDGE_MP_Solver,
          discount_factor=0.92)
```

EMERALD input data

Creating the JuDGE model

```
model = JuDGEModel(mytree,
          ConditionallyUniformProbabilities,
          sub_problems,
          JuDGE_MP_Solver,
          discount_factor=0.92)
```

EMERALD input data

Creating the JuDGE model

```
model = JuDGEModel(mytree,
          ConditionallyUniformProbabilities,
          sub_problems,
          JuDGE_MP_Solver,
          discount_factor=0.92)
```

EMERALD results

Solving and producing output

```
JuDGE.solve(model,termination=Termination(reltol=0.001))

resolve_subproblems(model)

solution = JuDGE.solution_to_dictionary(model)

(some code to set up custom_plots using plotly)

JuDGE.visualize_tree(mytree, solution,
custom=custom_plots)
```

EMERALD results

Solving and producing output

```
JuDGE.solve(model,termination=Termination(reltol=0.001))

resolve_subproblems(model)

solution = JuDGE.solution_to_dictionary(model)

(some code to set up custom_plots using plotly)

JuDGE.visualize_tree(mytree, solution,
custom=custom_plots)
```

EMERALD results

Solving and producing output

```
JuDGE.solve(model,termination=Termination(reltol=0.001))

resolve_subproblems(model)

solution = JuDGE.solution_to_dictionary(model)

(some code to set up custom_plots using plotly)

JuDGE.visualize_tree(mytree, solution,
custom=custom_plots)
```

EMERALD results

Solving and producing output

```
JuDGE.solve(model,termination=Termination(reltol=0.001))
resolve_subproblems(model)
solution = JuDGE.solution_to_dictionary(model)
(some code to set up custom_plots using plotly)
JuDGE.visualize_tree(mytree, solution,
custom=custom_plots)
```

The End

`JuDGE.jl` Julia Library downloadable from

`https://github.com/EPOC-NZ/JuDGE.jl`

My contact: `a.philpott@auckland.ac.nz`
Technical questions to: `a.downward@auckland.ac.nz`