Applying Multi-stage Stochastic Programming to Investment Planning for Energy Systems

A. Downward, A.B. Philpott

Electric Power Optimization Centre

Department of Engineering Science,
University of Auckland

INFORMS Annual Meeting 2019, Seattle WA.

Outline

Outline

New Zealand Interim Climate Change Committee

Home / What we do / Energy

Energy

> New Zealand has set a target under the Paris Agreement to reduce its greenhouse gas emissions by 30% below 2005 levels by 2030, and to adopt increasingly more ambitious targets in the future.

The Government has asked us to deliver evidence and analysis to a proposed Climate Change Commission. Regarding electricity, the Commission will be expected to make recommendations on:

Planning for the transition to 100% renewable electricity by 2035 (which includes geothermal) in a normal hydrological year.

DEEP DIVE

California strives to nix its natural gas habit without letting the lights go out

The state relies on natural gas for reliability, but won't reach its 100% zero emissions goal without looking to alternatives.

AUTHOR
Herman K. Trabish

When California's lawmakers mandated 60% renewables by 2030 and targeted zero emissions by 2045 in Senate Bill 100, passed in September, they took on never-before-answered questions about reliability.

https://www.utilitydive.com

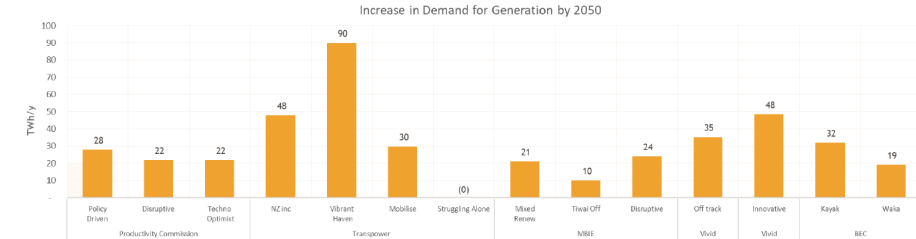
What does 100% renewable mean?

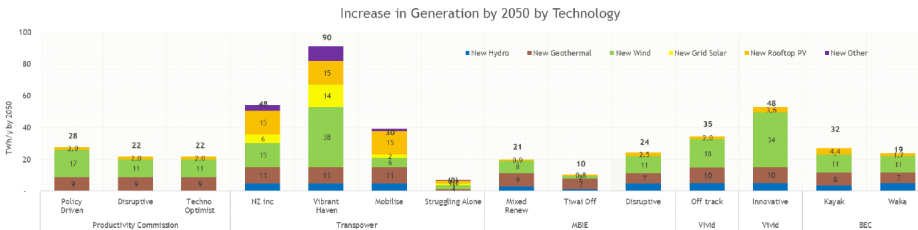
- Permanently shutdown thermal plant?
- "100% in a normal hydrological year."
- Keep thermal units, but use only in a low-hydrology year?
  - e.g. ex-post identification of a "normal" year;
  - using thermal plant in at most 50% of historical years is very expensive.
- Real aim of climate policy is to control average GHG emissions from electricity generation to below an accepted threshold.
- Planning for future years involves uncertainty, so we need stochastic models which account for:
  - uncertain long-term demand growth e.g. in electric vehicles;
  - the fact that capacity plans that affect hydro security of supply.

Outline

Scenarios for the future

What investments should be made?



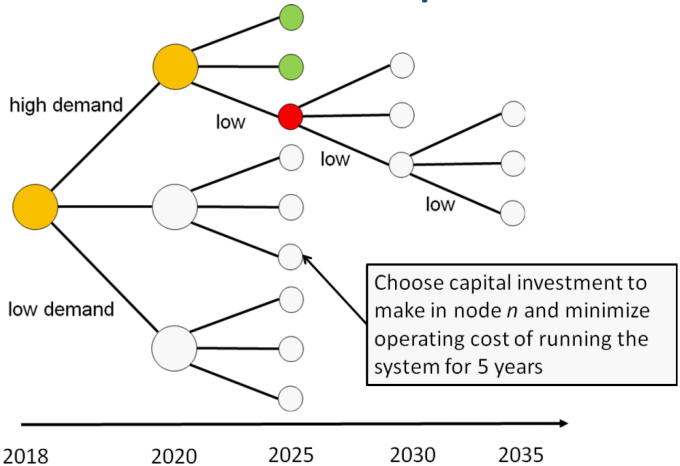


14 scenarios for electricity demand and generation mix in 2050.

There are 14 different ‘optimal’ plans: which should be implemented, if any?

Multi-stage stochastic investment problem

Scenario tree



Options to delay investment when in the yellow nodes until more information accrues. In green nodes invest and in red node don't. We call this multistage model GEMstone.[2]

[2] Girardeau P. & Philpott A.B., GEMSTONE: a stochastic GEM (2011) http://www.epoc.org.nz/workshops/ww2011/WW2011-girardeau.pdf.

Multi-stage stochastic investment problem

EMERALD

[3]Downward A. & Philpott A.B. Average-cost infinite-horizon SDDP (2019).

Multi-stage stochastic investment problem

EMERALD

- Emissions Mitigation for Electricity with Reservoir Approximation and Long-term Demand growth.

[3] Downward A. & Philpott A.B. Average-cost infinite-horizon SDDP (2019).

Multi-stage stochastic investment problem

EMERALD

- Emissions Mitigation for Electricity with Reservoir Approximation and Long-term Demand growth.
- Ideally, for each node in the tree, once we choose the generation and transmission investments, we would find a hydro release policy which minimizes expected variable costs per year of meeting demand (with possible load shedding).
- State of the art algorithm for hydro-thermal optimization is Stochastic Dual Dynamic Programming (SDDP). [Pereira and Pinto, 1991]
- SDDP optimizes a stochastic control problem by estimating the value of water at each stage, given the the expected future cost of meeting demand.

[3] Downward A. & Philpott A.B. Average-cost infinite-horizon SDDP (2019).

Multi-stage stochastic investment problem

EMERALD

- Emissions Mitigation for Electricity with Reservoir Approximation and Long-term Demand growth.
- Ideally, for each node in the tree, once we choose the generation and transmission investments, we would find a hydro release policy which minimizes expected variable costs per year of meeting demand (with possible load shedding).
- State of the art algorithm for hydro-thermal optimization is Stochastic Dual Dynamic Programming (SDDP). [Pereira and Pinto, 1991]
- SDDP optimizes a stochastic control problem by estimating the value of water at each stage, given the the expected future cost of meeting demand.
- We need a version of SDDP that minimizes average cost per year, say $\mathbb{E}[Z]$.[3]
- Unfortunately, it's computationally intractable to embed this within a stochastic program for investment.

[3] Downward A. & Philpott A.B. Average-cost infinite-horizon SDDP (2019).

Scenarios for the future

Hydrology Approximations

In order to maintain tractability in a long-term planning model, we consider four seasons of hydro inflows over a number of years.

We model two types of hydro plant: run-of-river and reservoir storage.

- For the run-of-river plants, we modify to available seasonal capacity (MW), based on the type of year.
- For reservoir storage, at the beginning of the year, we choose hydro storage targets for the end of each season, and ensure that regardless of the inflows, we reach these targets (either by increase or decreasing the output (MWh) from the plant over the season.

Multistage stochastic investment problem

Stage problem

$$
\begin{aligned}
Q_t(X_{t-1}, \omega) = \min \quad & \sum_{p \in \mathcal{P}} C^{p,\omega}(X_t^p - X_{t-1}^p) + \sum_{p \in \mathcal{P}} \sum_{h \in \mathcal{H}} c^{p,\omega} x_h^{p,\omega} + \rho \mathcal{Q}_{t+1}(X_{t+1}) \\
\text{subject to} \quad & X_t^p \in \mathcal{X}^p \\
& 0 \leq x_h^{p,\omega} \leq \alpha_h^p X_h \quad \forall p \in \mathcal{P}, h \in \mathcal{H} \\
& W_s^p - \sum_{h \in \mathcal{S}} x_h^{p,\omega} - G_s^p(\omega) \leq W_{s\%4+1}^p \quad \forall p \in P_H, s \in \{1,2,3,4\} \\
& \sum_{p \in \mathcal{P}} x_h^{p,\omega} = d_t(\omega) \quad \forall h \in \mathcal{H}
\end{aligned}
$$

where $\mathcal{Q}_{t+1}(X) = \mathbb{E}_\omega \left[Q_{t+1}(X_{t+1}, \omega) \right]$.

Multistage stochastic investment problem

Stage problem

$$
\begin{aligned}
\text{Cost-to-go}(t) = \min \quad & \text{Investment cost}(t) + \text{Operation Cost}(t) + \text{Cost-to-go}(t+1) \\
\text{subject to} \quad & \text{(Plant build capacity)} \\
& \text{(Plant availability by period)} \\
& \text{(Seasonal hydro restrictions)} \\
& \text{(Demand balance)}
\end{aligned}
$$

Multistage stochastic investment problem

JuDGE

Unfortunately, due to the lumpiness of investments in a small market such as New Zealand's, we cannot assume continuous investments, and must consider individual projects, and they will either be chosen to be built in a particular period or not. This involves a number of binary variables, meaning that SDDP.jl[4] cannot be utilised.

[4] Dowson O. & Kapelevich L. SDDP.jl: a Julia package for stochastic dual dynamic programming (2017).

[5] Baucke R., Downward A. & Philpott A.B. JuDGE.jl: a Julia package for optimizing capacity expansion

Multistage stochastic investment problem

JuDGE

Unfortunately, due to the lumpiness of investments in a small market such as New Zealand's, we cannot assume continuous investments, and must consider individual projects, and they will either be chosen to be built in a particular period or not. This involves a number of binary variables, meaning that SDDP.jl[4] cannot be utilised.

We have developed an alternative multi-stage stochastic programming engine in Julia, called JuDGE[5] (Julia Decomposition for General Expansion). This implements Dantzig-Wolfe decomposition on multistage investment problems described over trees.

[4] Dowson O. & Kapelevich L. SDDP.jl: a Julia package for stochastic dual dynamic programming (2017).

[5] Baucke R., Downward A. & Philpott A.B. JuDGE.jl: a Julia package for optimizing capacity expansion

Outline

JuDGE

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that enables easy construction of multi-stage stochastic capacity expansion problems.

JuDGE

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that enables easy construction of multi-stage stochastic capacity expansion problems.

- $\mathcal{N}$ 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 binary expansion variables;
- z_n are the binary variables for the expansions;
- y_n is the variable vector for stage-problem n;
- $\mathcal{Y}_n$ is the stage-problem feasibility set.

$$
\begin{aligned}
\min_{y,z} \quad & \sum_{n \in \mathcal{N}} \phi_n (c_n^\top z_n + q_n^\top y_n) \\
\text{s.t.} \quad & A_n y_n \leq b + D \sum_{h \in \mathcal{P}_n} z_h, \ \forall n \in \mathcal{N}, \\
& y_n \in \mathcal{Y}_n, \ \forall n \in \mathcal{N}, \\
& \sum_{h \in \mathcal{P}_n} z_h \leq \mathbf{1}^m, \ \forall n \in \mathcal{N}, \\
& z_n \in \{0,1\}^m, \ \forall n \in \mathcal{N}.
\end{aligned}
$$

JuDGE

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that enables easy construction of multi-stage stochastic capacity expansion problems.

$$
\begin{aligned}
\min_{y,z} \quad & \sum_{n \in \mathcal{N}} \phi_n (c_n^\top z_n + q_n^\top y_n) \\
\text{s.t.} \quad & A_n y_n \leq b + D \sum_{h \in \mathcal{P}_n} z_h, \ \forall n \in \mathcal{N}, \\
& y_n \in \mathcal{Y}_n, \ \forall n \in \mathcal{N}, \\
& \sum_{h \in \mathcal{P}_n} z_h \leq \mathbf{1}^m, \ \forall n \in \mathcal{N}, \\
& z_n \in \{0,1\}^m, \ \forall n \in \mathcal{N}.
\end{aligned}
$$

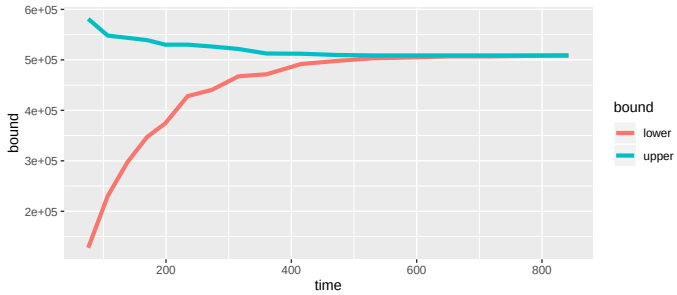
JuDGE implements a Dantzig-Wolfe decomposition of the problem by automatically constructing a master problem that handles the investment decisions, and generates columns from the stage-problems.

JuDGE

Benefits

JuDGE has several benefits compared to formulating a multi-stage stochastic investment problem directly.

- It separates the structure of the scenario tree from the stage-problem.
- Decomposition allows problems to be solved that would otherwise be too large.
- We still compute lower and upper bounds, like a MIP.



JuDGE

Interface

The interface to JuDGE involves:

- defining the tree, which enables custom data to be stored at each node (the tree can be custom-built, or specified in terms of depth, degree. To construct a binary tree you type:

```
mytree = buildtree(depth=5,degree=2)
m = JuDGEModel(mytree)
```

- Investments are defined as custom JuMP variables:

```
JuDGEexpansions!(m) do sp
  @expansion(sp,bags[1:2])
end
```

JuDGE

Interface

The interface to JuDGE involves:

- defining the tree, which enables custom data to be stored at each node (the tree can be custom-built, or specified in terms of depth, degree. To construct a binary tree you type:

```
mytree = buildtree(depth=5,degree=2)
m = JuDGEModel(mytree)
```

- Investments are defined as custom JuMP variables:

```
JuDGEexpansions!(m) do sp
  @expansion(sp,bags[1:2])
end
```

JuDGE

Interface

The interface to JuDGE involves:

- defining the tree, which enables custom data to be stored at each node (the tree can be custom-built, or specified in terms of depth, degree. To construct a binary tree you type:

```
mytree = buildtree(depth=5,degree=2)
m = JuDGEModel(mytree)
```

- Investments are defined as custom JuMP variables:

```
JuDGEexpansions!(m) do sp
  @expansion(sp,bags[1:2])
end
```

JuDGE

Interface

The interface to JuDGE involves:

- defining the tree, which enables custom data to be stored at each node (the tree can be custom-built, or specified in terms of depth, degree. To construct a binary tree you type:

```
mytree = buildtree(depth=5,degree=2)
m = JuDGEModel(mytree)
```

- Investments are defined as custom JuMP variables:

```
JuDGEexpansions!(m) do sp
   @expansion(sp,bags[1:2])
end
```

JuDGE

Interface

The interface to JuDGE involves:

- defining the tree, which enables custom data to be stored at each node (the tree can be custom-built, or specified in terms of depth, degree. To construct a binary tree you type:

```
mytree = buildtree(depth=5,degree=2)
m = JuDGEModel(mytree)
```

- Investments are defined as custom JuMP variables:

```
JuDGEexpansions!(m) do sp
  @expansion(sp,bags[1:2])
end
```

JuDGE

Interface

- the costs the investments are then defined:

```
JuDGEexpansioncosts!(m) do master,n,expansion
   bags = expansion[:bags]
   @expression(master,sum(n.data.cost[o]*bags[o] for o in 1:2))
end
```

- next, the stage-problem is defined as follows:

```
JuDGEsubproblem!(m) do sp,n,expansion
   bags = expansion[:bags]
   @variable(sp, y[1:10], category=:Bin)
   @objective(sp, Min, sum(-n.data.value[i]*y[i] for i in 1:10))
   @constraint(sp, sum(n.data.volume[i]*y[i] for i in 1:10) <=
      initialcap + sum(investvol[o]*bags[o] for o in 1:2))
end
```

JuDGE

Interface

- the costs the investments are then defined:

```
JuDGEexpansioncosts!(m) do master,n,expansion
  bags = expansion[:bags]
  @expression(master,sum(n.data.cost[o]*bags[o] for o in 1:2))
end
```

- next, the stage-problem is defined as follows:

```
JuDGEsubproblem!(m) do sp,n,expansion
  bags = expansion[:bags]
  @variable(sp, y[1:10], category=:Bin)
  @objective(sp, Min, sum(-n.data.value[i]*y[i] for i in 1:10))
  @constraint(sp, sum(n.data.volume[i]*y[i] for i in 1:10) <=
    initialcap + sum(investvol[o]*bags[o] for o in 1:2))
end
```

JuDGE

Interface

- the costs the investments are then defined:

```
JuDGEexpansioncosts!(m) do master,n,expansion
   bags = expansion[:bags]
   @expression(master,sum(n.data.cost[o]*bags[o] for o in 1:2))
end
```

- next, the stage-problem is defined as follows:

```
JuDGEsubproblem!(m) do sp,n,expansion
   bags = expansion[:bags]
   @variable(sp, y[1:10], category=:Bin)
   @objective(sp, Min, sum(-n.data.value[i]*y[i] for i in 1:10))
   @constraint(sp, sum(n.data.volume[i]*y[i] for i in 1:10) <=
      initialcap + sum(investvol[o]*bags[o] for o in 1:2))
end
```

JuDGE

Interface

- the costs the investments are then defined:

```
JuDGEexpansioncosts!(m) do master,n,expansion
   bags = expansion[:bags]
   @expression(master,sum(n.data.cost[o]*bags[o] for o in 1:2))
end
```

- next, the stage-problem is defined as follows:

```
JuDGEsubproblem!(m) do sp,n,expansion
   bags = expansion[:bags]
   @variable(sp, y[1:10], category=:Bin)
   @objective(sp, Min, sum(-n.data.value[i]*y[i] for i in 1:10))
   @constraint(sp, sum(n.data.volume[i]*y[i] for i in 1:10) <=
      initialcap + sum(investvol[o]*bags[o] for o in 1:2))
end
```

JuDGE

Interface

- the costs the investments are then defined:

```
JuDGEexpansioncosts!(m) do master,n,expansion
   bags = expansion[:bags]
   @expression(master,sum(n.data.cost[o]*bags[o] for o in 1:2))
end
```

- next, the stage-problem is defined as follows:

```
JuDGEsubproblem!(m) do sp,n,expansion
   bags = expansion[:bags]
   @variable(sp, y[1:10], category=:Bin)
   @objective(sp, Min, sum(-n.data.value[i]*y[i] for i in 1:10))
   @constraint(sp, sum(n.data.volume[i]*y[i] for i in 1:10) <=
      initialcap + sum(investvol[o]*bags[o] for o in 1:2))
end
```

JuDGE

Interface

- the costs the investments are then defined:

```
JuDGEexpansioncosts!(m) do master,n,expansion
   bags = expansion[:bags]
   @expression(master,sum(n.data.cost[o]*bags[o] for o in 1:2))
end
```

- next, the stage-problem is defined as follows:

```
JuDGEsubproblem!(m) do sp,n,expansion
   bags = expansion[:bags]
   @variable(sp, y[1:10], category=:Bin)
   @objective(sp, Min, sum(-n.data.value[i]*y[i] for i in 1:10))
   @constraint(sp, sum(n.data.volume[i]*y[i] for i in 1:10) <=
      initialcap + sum(investvol[o]*bags[o] for o in 1:2))
end
```

JuDGE

Interface

- the costs the investments are then defined:

```
JuDGEexpansioncosts!(m) do master,n,expansion
   bags = expansion[:bags]
   @expression(master,sum(n.data.cost[o]*bags[o] for o in 1:2))
end
```

- next, the stage-problem is defined as follows:

```
JuDGEsubproblem!(m) do sp,n,expansion
   bags = expansion[:bags]
   @variable(sp, y[1:10], category=:Bin)
   @objective(sp, Min, sum(-n.data.value[i]*y[i] for i in 1:10))
   @constraint(sp, sum(n.data.volume[i]*y[i] for i in 1:10) <=
      initialcap + sum(investvol[o]*bags[o] for o in 1:2))
end
```

JuDGE

Interface

- the costs the investments are then defined:

```
JuDGEexpansioncosts!(m) do master,n,expansion
  bags = expansion[:bags]
  @expression(master,sum(n.data.cost[o]*bags[o] for o in 1:2))
end
```

- next, the stage-problem is defined as follows:

```
JuDGEsubproblem!(m) do sp,n,expansion
  bags = expansion[:bags]
  @variable(sp, y[1:10], category=:Bin)
  @objective(sp, Min, sum(-n.data.value[i]*y[i] for i in 1:10))
  @constraint(sp, sum(n.data.volume[i]*y[i] for i in 1:10) <=
    initialcap + sum(investvol[o]*bags[o] for o in 1:2))
end
```

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

- finally we solve the problem:

```
JuDGEsolve!(m,GurobiSolver(OutputFlag=0)) do time, iter, lb, ub
  println(lb," ",ub," ",time)
  if time > 10 || iter > 200
    return true
  end
  if (ub - lb) < 0.001
    return true
  end
  return false
end
```

- finally we solve the problem:

```
JuDGEsolve!(m,GurobiSolver(OutputFlag=0)) do time, iter, lb, ub
  println(lb," ",ub," ",time)
  if time > 10 || iter > 200
    return true
  end
  if (ub - lb) < 0.001
    return true
  end
  return false
end
```

- finally we solve the problem:

```
JuDGEsolve!(m,GurobiSolver(OutputFlag=0)) do time, iter, lb, ub
  println(lb," ",ub," ",time)
  if time > 10 || iter > 200
    return true
  end
  if (ub - lb) < 0.001
    return true
  end
  return false
end
```

Outline

New Zealand Electricity Market Overview

Generation Locations

NON-RENEWABLE

FOSSIL FUELS
Includes coal, oil and natural gas. The energy comes from the fossilised remains of plants and animals from millions of years ago.

RENEWABLE

HYDRO ENERGY
Energy created by falling water

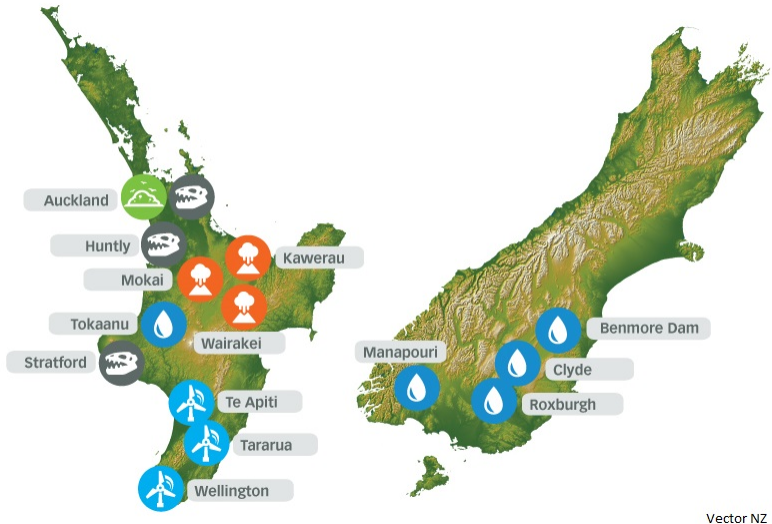
WIND ENERGY
Energy from the force of wind

GEOTHERMAL ENERGY
Energy from underground steam

SOLAR ENERGY
Energy from the sun

BIOENERGY
Fuel/Energy from waste materials

MARINE ENERGY
Energy from tidal movements and waves



Vector NZ

Case-study

Assumptions

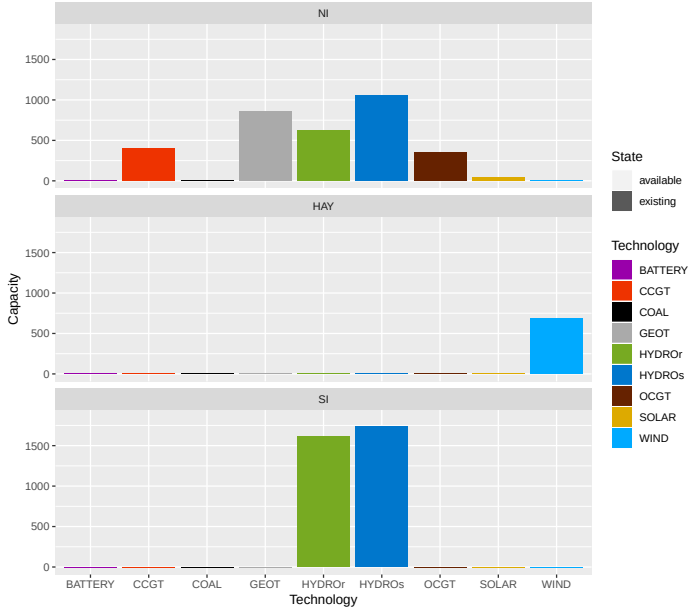
We will now present a case-study inspired by New Zealand.

- We model New Zealand as three nodes.
- We will explore both carbon-price trajectories, and explicit restrictions on non-renewables.
- We use 30 load blocks / wind scenarios, for each of the four seasons.
- We use 13 historical years to calibrate seasonal hydrological inflows.
- Data is based on Michael Ferris and Andy Philpott's two-stage renewables model.[6]

[6] Ferris M.C. & Philpott A.B. 100% Renewables with Storage (2019) `http://www.epoc.org.nz/papers/100PercentOperResv10.pdf`.

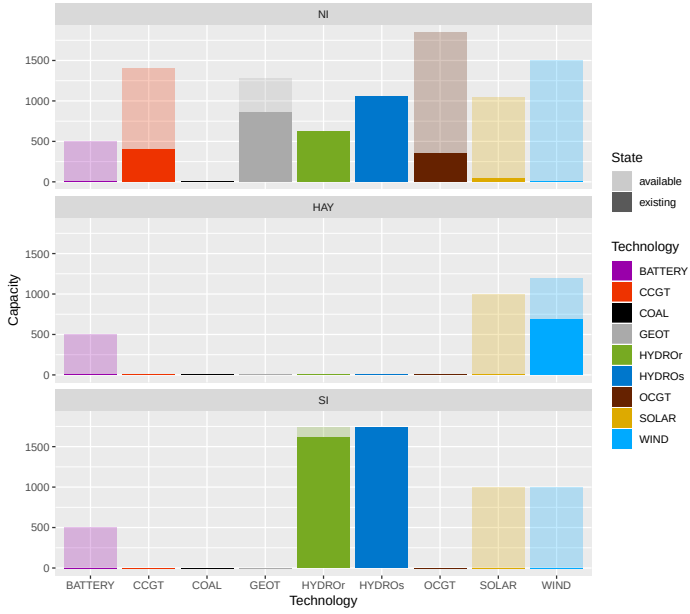
Model Inputs

Technologies



Model Inputs

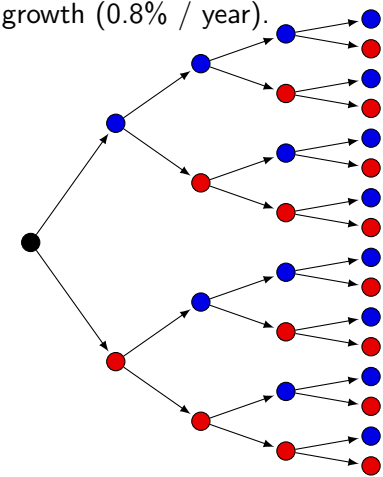
Technologies



Model Inputs

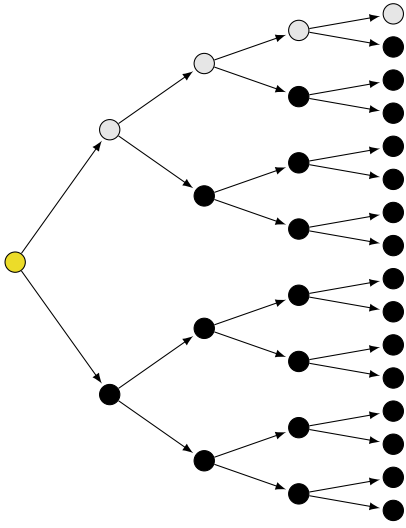
Scenario Tree

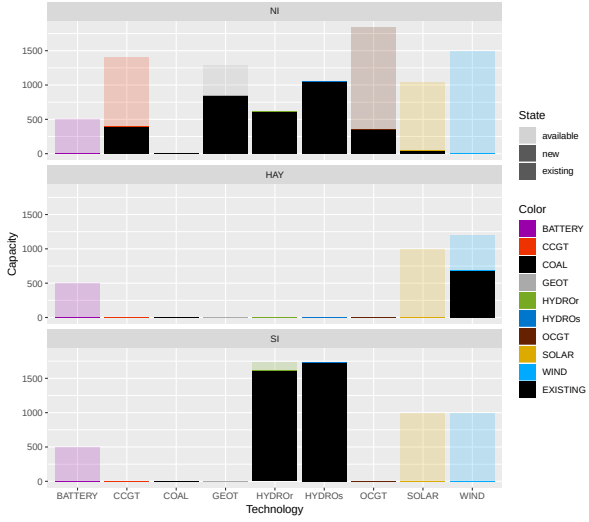
In this tree, we model 2020–2050; each node represents 6 years of the electricity system. Blue nodes correspond to high demand growth (1.6% / year), and red nodes correspond to low demand growth (0.8% / year).



Results

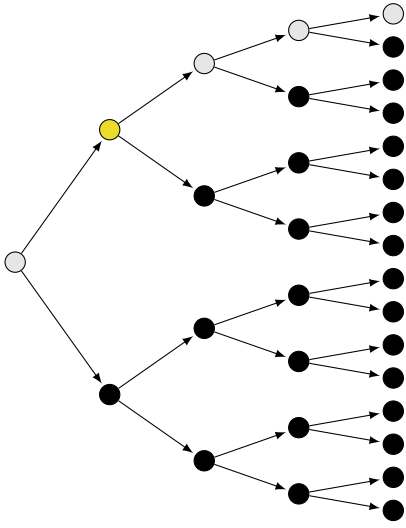
High Demand Trajectory

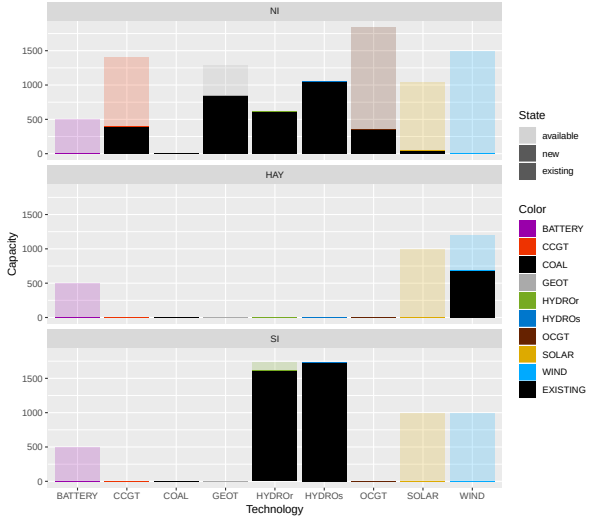




Results

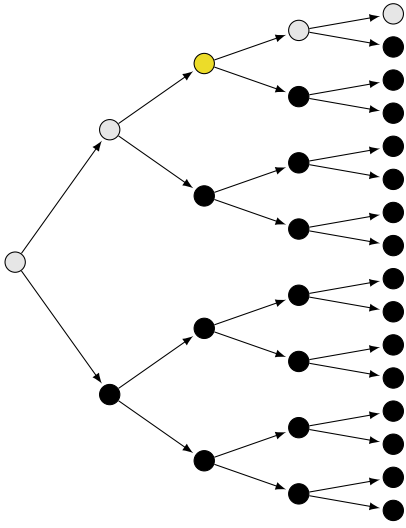
High Demand Trajectory

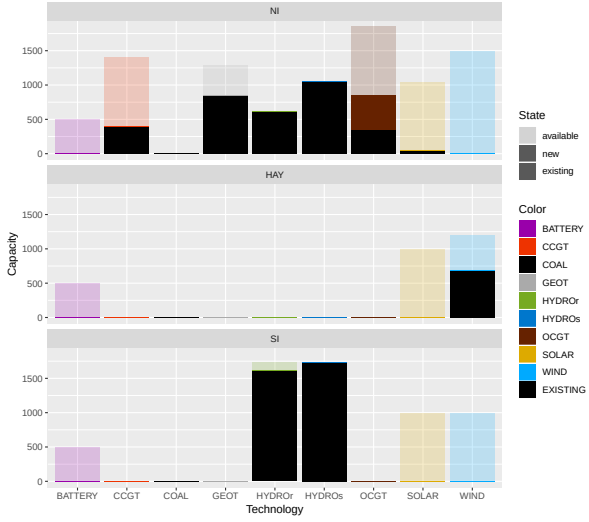




Results

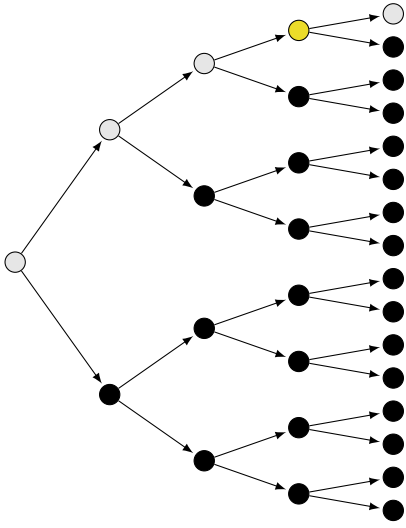
High Demand Trajectory

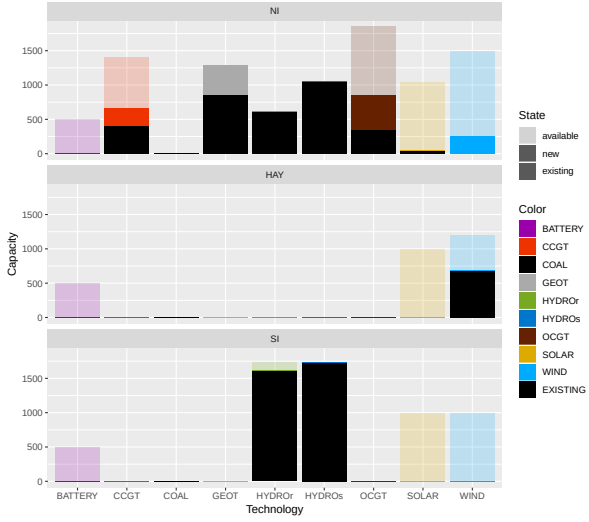




Results

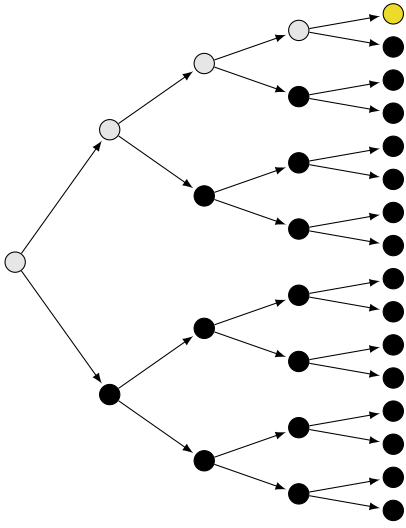
High Demand Trajectory

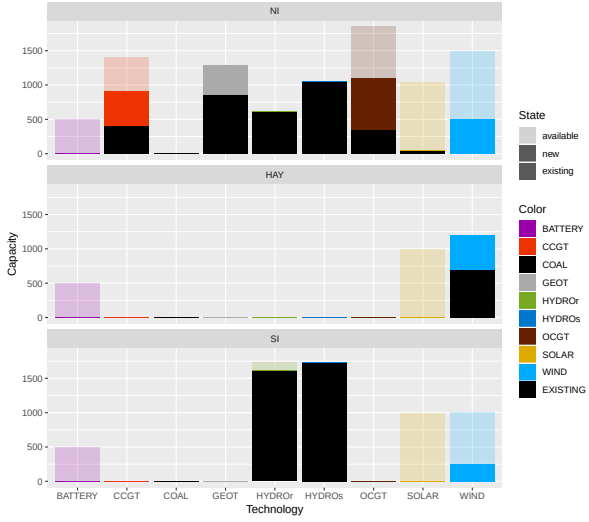




Results

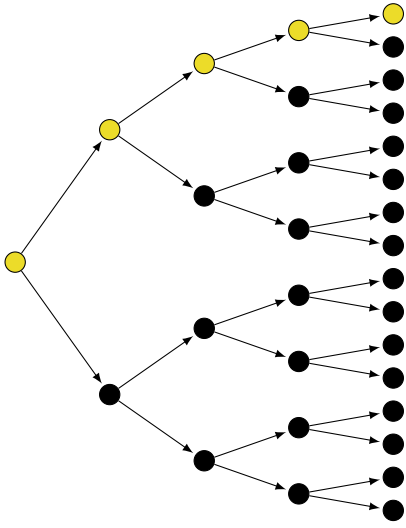
High Demand Trajectory

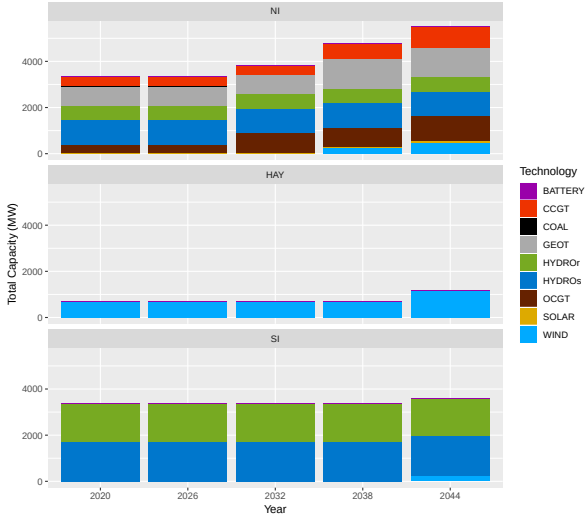




Results

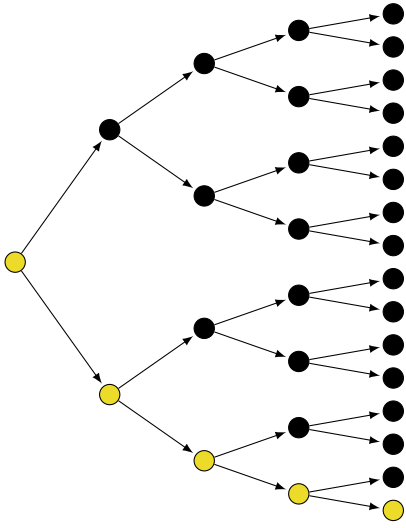
High Demand Trajectory

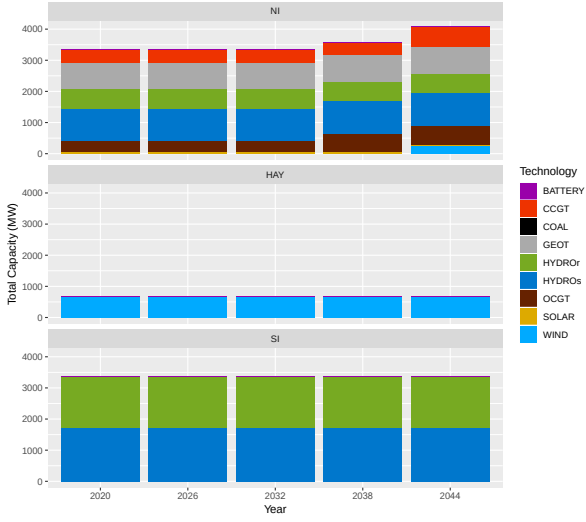




Results

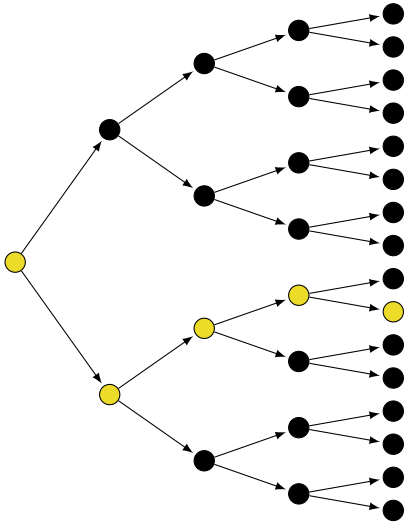
Low Demand Trajectory

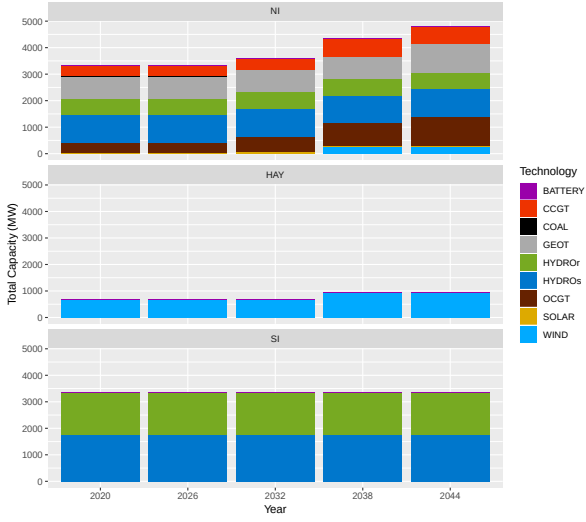




Results

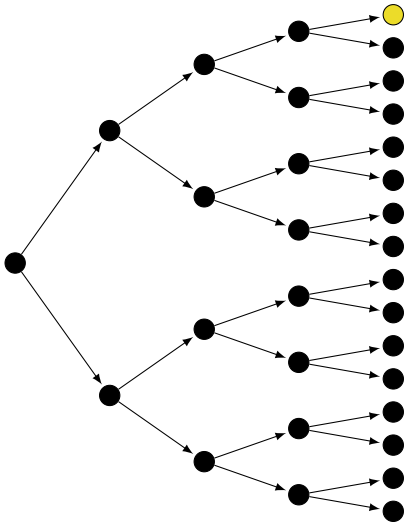
Medium Demand Trajectory

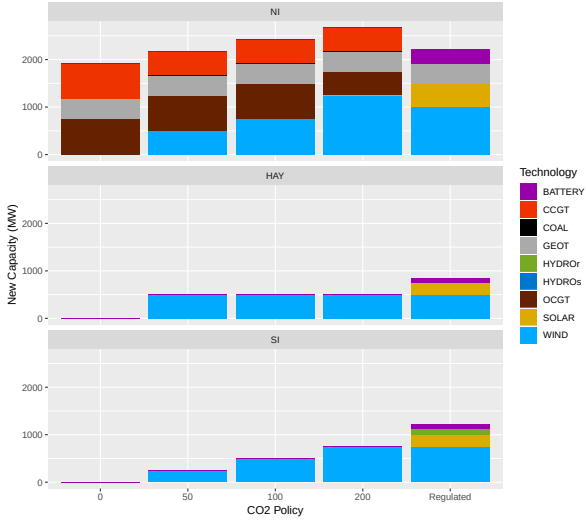




Results

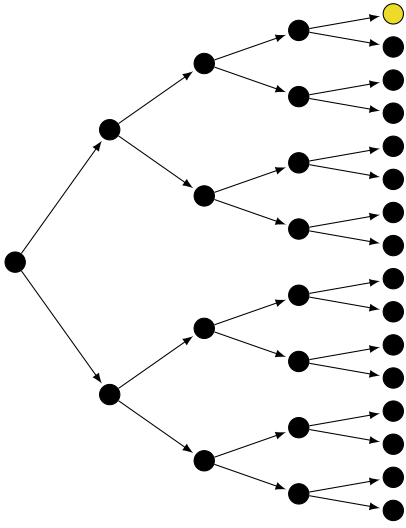
Carbon Policy Comparison (Investments)

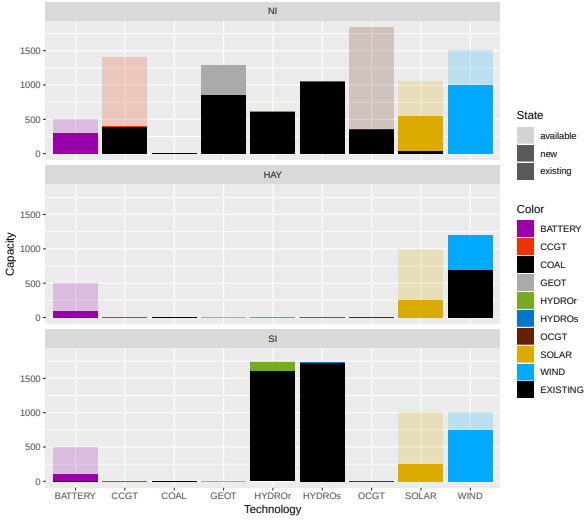




Results

Carbon Policy: 100% Renewables





Outline

Conclusions

Summary

- We have developed a Julia package, JuDGE, that enables easy modelling of multistage stochastic binary capacity expansion problems, with mixed-integer stage-problems.
- We have applied JuDGE to the problem of decarbonising New Zealand's electricity sector, creating a model called EMERALD.
- We find that directly regulating the use of non-renewable technologies leads to the deployment of solar and battery technology, which would not be optimal with reasonable prices for CO_2 emissions.

Conclusions

Summary

- We have developed a Julia package, JuDGE, that enables easy modelling of multistage stochastic binary capacity expansion problems, with mixed-integer stage-problems.
- We have applied JuDGE to the problem of decarbonising New Zealand's electricity sector, creating a model called EMERALD.
- We find that directly regulating the use of non-renewable technologies leads to the deployment of solar and battery technology, which would not be optimal with reasonable prices for CO_2 emissions.
- We plan to continue to make enhancements to JuDGE; including: upgrading to Julia 1.*, adding multi-threading, and extending the types of models that can be solved.
- We need to further calibrate the EMERALD model enhancing the detail captured in the stage-problem, and the size of the tree.

Thanks for your attention.

Any questions?

JuDGE.jl Julia Library https://github.com/reganbaucke/JuDGE.jl

Contact me: a.downward@auckland.ac.nz