Multistage stochastic capacity planning in networks

Anthony Downward

Joint work with Andy Philpott

Electric Power Optimization Centre
Engineering Science
University of Auckland

31st European Conference on Operational Research, 14 July 2021.

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

Outline

Outline

Background

- Multi-horizon stochastic programming
- Applications for network models
- Implementation and communication of policies

JuDGE: Julia-based Decomposition for General Expansion

Simple Capacity Planning Model

Implementation using JuDGE

Computational Benchmarks

Communication of JuDGE Solutions

Background

Multi-horizon planning

The concept of multi-horizon modelling is the idea that you are planning for across multiple time-horizons at once (short-term, medium-term, and long-term) and are explicitly accounting for how strategic, tactical and operational decisions influence each other.

Background

Multi-horizon planning

The concept of multi-horizon modelling is the idea that you are planning for across multiple time-horizons at once (short-term, medium-term, and long-term) and are explicitly accounting for how strategic, tactical and operational decisions influence each other.

In this talk, we will be considering this type of problem in the context of stochastic capacity expansion models.

Background

Multi-horizon planning

The concept of multi-horizon modelling is the idea that you are planning for across multiple time-horizons at once (short-term, medium-term, and long-term) and are explicitly accounting for how strategic, tactical and operational decisions influence each other.

In this talk, we will be considering this type of problem in the context of stochastic capacity expansion models.

In the short-term, we have operational decisions that result in immediate costs and revenue; however, at the same time the decision maker is considering capacity expansion decisions that will lead to lower operational costs, or higher revenue in the future.

Background

Multi-horizon network models

A wide-range of network models can form the operational model for this class of problem, enabling the consequences of long-term investment decisions to be endogenously modelled.

- Water network planning for drought management: storage; water treatment; desalination.

Background

Multi-horizon network models

A wide-range of network models can form the operational model for this class of problem, enabling the consequences of long-term investment decisions to be endogenously modelled.

- Water network planning for drought management: storage; water treatment; desalination.
- Integration of renewables for electricity systems: transmission grid upgrades; location of new generation.

Background

Multi-horizon network models

A wide-range of network models can form the operational model for this class of problem, enabling the consequences of long-term investment decisions to be endogenously modelled.

- Water network planning for drought management: storage; water treatment; desalination.
- Integration of renewables for electricity systems: transmission grid upgrades; location of new generation.
- Electricity distribution network reliability: line upgrades; battery storage.

Background

Multi-horizon network models

A wide-range of network models can form the operational model for this class of problem, enabling the consequences of long-term investment decisions to be endogenously modelled.

- Water network planning for drought management: storage; water treatment; desalination.
- Integration of renewables for electricity systems: transmission grid upgrades; location of new generation.
- Electricity distribution network reliability: line upgrades; battery storage.
- Energy transition for industry: changes to the energy supply chain; plant modifications.

Background

Implementation and communication of policies

Stochastic programming has been promoted in academia for decades, but has only recently been gaining traction in capacity planning settings within business.

[2] M. Haasnoot, J.H. Kwakkel, W.E. Walker, J. ter Maat, Dynamic adaptive policy pathways: A method for crafting robust decisions for a deeply uncertain world (2013).

Background

Implementation and communication of policies

Stochastic programming has been promoted in academia for decades, but has only recently been gaining traction in capacity planning settings within business.

In part, the delay in acceptance has been due to the limitations in the size of problems that could be modelled, but more significantly the solution to a stochastic program is a policy that adapts to information as it is revealed, and this has been difficult communicate to decision makers.

[2] M. Haasnoot, J.H. Kwakkel, W.E. Walker, J. ter Maat, Dynamic adaptive policy pathways: A method for crafting robust decisions for a deeply uncertain world (2013).

Background

Implementation and communication of policies

Stochastic programming has been promoted in academia for decades, but has only recently been gaining traction in capacity planning settings within business.

In part, the delay in acceptance has been due to the limitations in the size of problems that could be modelled, but more significantly the solution to a stochastic program is a policy that adapts to information as it is revealed, and this has been difficult communicate to decision makers.

The concept of Dynamic Adaptive Pathways has made in-roads in areas where there is deep uncertainty, particularly climate change planning.[2] This, however, is typically more qualitative than quantitative.

[2]M. Haasnoot, J.H. Kwakkel, W.E. Walker, J. ter Maat, Dynamic adaptive policy pathways: A method for crafting robust decisions for a deeply uncertain world (2013).

Outline

Background

- Multi-horizon stochastic programming
- Applications for network models
- Implementation and communication of policies

JuDGE: Julia-based Decomposition for General Expansion

Simple Capacity Planning Model

Implementation using JuDGE

Computational Benchmarks

Communication of JuDGE Solutions

JuDGE Framework Overview

Implementation and communication of policies

In this talk, I will present JuDGE.jl[3], a Julia package which

- allows users to easily implement multi-horizon optimization models using the JuMP modelling language;

[3] A. Downward, R. Baucke, A.B. Philpott, JuDGE.jl: a Julia package for optimizing capacity expansion (2020).

JuDGE Framework Overview

Implementation and communication of policies

In this talk, I will present JuDGE.jl[3], a Julia package which

- allows users to easily implement multi-horizon optimization models using the JuMP modelling language;
- applies Dantzig-Wolfe decomposition in order to solve large-scale models;

[3] A. Downward, R. Baucke, A.B. Philpott, JuDGE.jl: a Julia package for optimizing capacity expansion (2020).

JuDGE Framework Overview

Implementation and communication of policies

In this talk, I will present JuDGE.jl[3], a Julia package which

- allows users to easily implement multi-horizon optimization models using the JuMP modelling language;
- applies Dantzig-Wolfe decomposition in order to solve large-scale models; and
- outputs an interactive view of the results over the scenario tree, enabling decision makers explore the optimal policy.

JuDGE stands for Julia Decomposition for Generalized Expansion.

[3] A. Downward, R. Baucke, A.B. Philpott, JuDGE.jl: a Julia package for optimizing capacity expansion (2020).

JuDGE Framework Overview

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that facilitates the modelling of multi-horizon stochastic capacity expansion problems.

JuDGE Framework Overview

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that facilitates the modelling of multi-horizon 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 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.

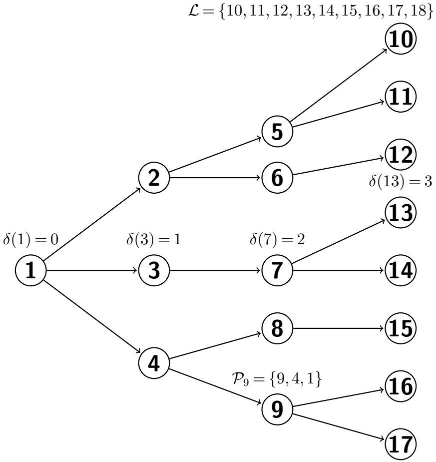
Extensive Form:

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

JuDGE Framework Overview

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that facilitates the modelling of multi-horizon stochastic capacity expansion problems.



Extensive Form:

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

JuDGE Framework Overview

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that facilitates the modelling of multi-horizon stochastic capacity expansion problems.

JuDGE applies Dantzig-Wolfe decomposition to the problem by automatically constructing a master problem that handles the investment decisions, and generates columns from the nodal subproblems.

These columns' costs are the operational costs of the nodal subproblems, and the columns' coefficients are the utilized investments.

Extensive Form:

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

JuDGE Framework Overview

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that facilitates the modelling of multi-horizon stochastic capacity expansion problems.

Restricted Master Problem:

The columns are indexed $j \in \mathcal{J}_n$ for each node n, and added to the restricted master problem, with cost ψ_n^j and coefficients $\hat{z}_n^j$.

$$
\begin{aligned}
\min_{x,w} \quad & \sum_{n \in \mathcal{N}} \phi_n \left(c_n^\top x_n + \sum_{j \in \mathcal{J}_n} \psi_n^j w_n^j \right) \\
\text{s.t.} \quad & \sum_{j \in \mathcal{J}_n} \hat{z}_n^j w_n^j \leq \sum_{h \in \mathcal{P}_n} x_h, \quad \forall n \in \mathcal{N}, \\
& \sum_{j \in \mathcal{J}_n} w_n^j = 1, \qquad \forall n \in \mathcal{N}, \\
& w_n^j, x_n \geq 0, \quad \forall n \in \mathcal{N}, j \in \mathcal{J}_n.
\end{aligned}
$$

This problem seeks to choose investments x that minimize the total expected cost, given the columns that have been generated.

JuDGE Framework Overview

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that facilitates the modelling of multi-horizon stochastic capacity expansion problems.

Restricted Master Problem:

The columns are indexed $j \in \mathcal{J}_n$ for each node n, and added to the restricted master problem, with cost ψ_n^j and coefficients $\hat{z}_n^j$.

$$
\begin{aligned}
\min_{x,w} \quad & \sum_{n\in\mathcal{N}} \phi_n \Big(c_n^\top x_n + \sum_{j\in\mathcal{J}_n} \psi_n^j w_n^j \Big) \\
\text{s.t.} \quad & \sum_{j\in\mathcal{J}_n} \hat{z}_n^j w_n^j \leq \sum_{h\in\mathcal{P}_n} x_h, \quad \forall n \in \mathcal{N}, \\
& \sum_{j\in\mathcal{J}_n} w_n^j = 1, \qquad \forall n \in \mathcal{N}, \\
& w_n^j, x_n \geq 0, \quad \forall n \in \mathcal{N}, j \in \mathcal{J}_n.
\end{aligned}
$$

This problem seeks to choose investments x that minimize the total expected cost, given the columns that have been generated.

JuDGE Framework Overview

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that facilitates the modelling of multi-horizon stochastic capacity expansion problems.

Restricted Master Problem:

The columns are indexed $j \in \mathcal{J}_n$ for each node n, and added to the restricted master problem, with cost ψ_n^j and coefficients $\hat{z}_n^j$.

$$
\begin{aligned}
\min_{x,w} \quad & \sum_{n \in \mathcal{N}} \phi_n \Big(c_n^\top x_n + \sum_{j \in \mathcal{J}_n} \psi_n^j w_n^j \Big) \\
\text{s.t.} \quad & \sum_{j \in \mathcal{J}_n} \hat{z}_n^j w_n^j \le \sum_{h \in \mathcal{P}_n} x_h, \quad \forall n \in \mathcal{N}, \\
& \sum_{j \in \mathcal{J}_n} w_n^j = 1, \quad \forall n \in \mathcal{N}, \\
& w_n^j, x_n \ge 0, \quad \forall n \in \mathcal{N}, j \in \mathcal{J}_n.
\end{aligned}
$$

This problem seeks to choose investments x that minimize the total expected cost, given the columns that have been generated. (Additional investments cannot decrease the set of feasible columns.)

JuDGE Framework Overview

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that facilitates the modelling of multi-horizon stochastic capacity expansion problems.

Restricted Master Problem:

$$
\begin{aligned}
\min_{x,w} \quad & \sum_{n\in\mathcal{N}} \phi_n\Big(c_n^\top x_n + \sum_{j\in\mathcal{J}_n} \psi_n^j w_n^j\Big) \\
\text{s.t.} \quad & \sum_{j\in\mathcal{J}_n} \hat{z}_n^j w_n^j \leq \sum_{h\in\mathcal{P}_n} x_h, \quad \forall n \in \mathcal{N}, \\
& \sum_{j\in\mathcal{J}_n} w_n^j = 1, \qquad\qquad \forall n \in \mathcal{N}, \\
& w_n^j, x_n \geq 0, \quad \forall n \in \mathcal{N}, j \in \mathcal{J}_n.
\end{aligned}
$$

This problem is solved without any integer variable restrictions, since dual variables are needed for the column generation process.

If the optimal solution is not naturally integer, JuDGE supports both MIP solves for the master, and branch-and-price to find integer feasible solutions.

JuDGE Framework Overview

What type of problems can be modelled in JuDGE?

JuDGE is a Julia/JuMP-based package that facilitates the modelling of multi-horizon stochastic capacity expansion problems.

Restricted Master Problem:

This problem is solved without any integer variable restrictions, since dual variables are needed for the column generation process.

$$
\begin{aligned}
\min_{x,w} \quad & \sum_{n\in\mathcal{N}} \phi_n\Big(c_n^\top x_n + \sum_{j\in\mathcal{J}_n} \psi_n^j w_n^j\Big) \\
\text{s.t.} \quad & \sum_{j\in\mathcal{J}_n} \hat{z}_n^j w_n^j \le \sum_{h\in\mathcal{P}_n} x_h, \ \forall n\in\mathcal{N}, \\
& \sum_{j\in\mathcal{J}_n} w_n^j = 1, \qquad \forall n\in\mathcal{N}, \\
& w_n^j, x_n \ge 0, \quad \forall n\in\mathcal{N}, j\in\mathcal{J}_n.
\end{aligned}
$$

If the optimal solution is not naturally integer, JuDGE supports both MIP solves for the master, and branch-and-price to find integer feasible solutions.

JuDGE Framework Overview

Elements of a JuDGE Model

JuDGE enables the formulation of multistage stochastic capacity management problems leveraging the JuMP mathematical modelling language within Julia.

JuDGE Framework Overview

Elements of a JuDGE Model

JuDGE enables the formulation of multistage stochastic capacity management problems leveraging the JuMP mathematical modelling language within Julia.

Modelling these problems consists of several elements:

- 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.

JuDGE Framework Overview

Elements of a JuDGE Model

JuDGE enables the formulation of multistage stochastic capacity management problems leveraging the JuMP mathematical modelling language within Julia.

Modelling these problems consists of several elements:

- 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.

JuDGE Framework Overview

Elements of a JuDGE Model

JuDGE enables the formulation of multistage stochastic capacity management problems leveraging the JuMP mathematical modelling language within Julia.

Modelling these problems consists of several elements:

- 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.

JuDGE Framework Overview

Elements of a JuDGE Model

JuDGE enables the formulation of multistage stochastic capacity management problems leveraging the JuMP mathematical modelling language within Julia.

Modelling these problems consists of several elements:

- 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.

JuDGE Framework Overview

Elements of a JuDGE Model

JuDGE enables the formulation of multistage stochastic capacity management problems leveraging the JuMP mathematical modelling language within Julia.

Modelling these problems consists of several elements:

- 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.

Given these elements, JuDGE can automatically form the restricted master problem, and provide the machinery necessary for the iterations of the Dantzig-Wolfe algorithm.

JuDGE Framework Overview

Elements of a JuDGE Model

JuDGE enables the formulation of multistage stochastic capacity management problems leveraging the JuMP mathematical modelling language within Julia.

Modelling these problems consists of several elements:

- 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.

Given these elements, JuDGE can automatically form the restricted master problem, and provide the machinery necessary for the iterations of the Dantzig-Wolfe algorithm.

Various solvers can be specified for the different models that are being solved. 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 Framework Overview

Elements of a JuDGE Model

JuDGE enables the formulation of multistage stochastic capacity management problems leveraging the JuMP mathematical modelling language within Julia.

Modelling these problems consists of several elements:

- 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.

Given these elements, JuDGE can automatically form the restricted master problem, and provide the machinery necessary for the iterations of the Dantzig-Wolfe algorithm.

Various solvers can be specified for the different models that are being solved. 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.

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

Outline

Facility Expansion Planning Model

Defining the Subproblems

We will now consider a simple multistage facility expansion problem, and go through the steps necessary to model this.

In our earlier formulation the nodal subproblem was simply written as: $y_n \in \mathcal{Y}_n$; let's now define it fully for this example.

Facility Expansion Planning Model

Defining the Subproblems

We will now consider a simple multistage facility expansion problem, and go through the steps necessary to model this.

We have sets / indices:

- supplies $i \in \mathcal{S}$; demands $j \in \mathcal{D}$; and routes $ij \in \mathcal{S} \times \mathcal{D}$.

Facility Expansion Planning Model

Defining the Subproblems

We will now consider a simple multistage facility expansion problem, and go through the steps necessary to model this.

We have sets / indices:

- supplies $i \in \mathcal{S}$; demands $j \in \mathcal{D}$; and routes $ij \in \mathcal{S} \times \mathcal{D}$.

The variables are:

- f_{ij} the flow on route ij; and
- x_i the number of upgrades of supply i.

Facility Expansion Planning Model

Defining the Subproblems

We will now consider a simple multistage facility expansion problem, and go through the steps necessary to model this.

We have sets / indices:

- supplies $i \in \mathcal{S}$; demands $j \in \mathcal{D}$; and routes $ij \in \mathcal{S} \times \mathcal{D}$.

The variables are:

- f_{ij} the flow on route ij; and
- x_i the number of upgrades of supply i.

The parameters are:

- c_{ij} per-unit cost of flow on route ij;
- C_i cost of increasing the capacity of supply i;
- s_i initial capacity of supply i; and
- S_i increase in capacity of supply i for each upgrade.

Facility Expansion Planning Model

Defining the Subproblems

Supply constraints:

$$\sum_{j \in \mathcal{D}} f_{ij} \leq s_i + S_i x_i, \quad \forall i \in \mathcal{S},$$

Demand constraints:

$$\sum_{i \in \mathcal{S}} f_{ij} = d_j, \quad \forall j \in \mathcal{D},$$

Binary expansions:

$$x_i \in \mathcal{Z}_+ \quad \forall i \in \mathcal{S}.$$

Objective function:

$$\min \sum_{ij \in \mathcal{S} \times \mathcal{D}} c_{ij} f_{ij} + \sum_{i \in \mathcal{S}} C_i x_i.$$

Facility Expansion Planning Model

Defining the Subproblems

Supply constraints:

$$\sum_{j \in \mathcal{D}} f_{ij} \le s_i + S_i x_i, \quad \forall i \in \mathcal{S},$$

Demand constraints:

$$\sum_{i \in \mathcal{S}} f_{ij} = d_j, \quad \forall j \in \mathcal{D},$$

Binary expansions:

$$x_i \in \mathcal{Z}_+ \quad \forall i \in \mathcal{S}.$$

Objective function:

$$\min \sum_{ij \in \mathcal{S} \times \mathcal{D}} c_{ij} f_{ij} + \sum_{i \in \mathcal{S}} C_i x_i.$$

Facility Expansion Planning Model

Defining the Subproblems

Supply constraints:

$$\sum_{j \in \mathcal{D}} f_{ij} \leq s_i + S_i x_i, \quad \forall i \in \mathcal{S},$$

Demand constraints:

$$\sum_{i \in \mathcal{S}} f_{ij} = d_j, \quad \forall j \in \mathcal{D},$$

Binary expansions:

$$x_i \in \mathcal{Z}_+ \quad \forall i \in \mathcal{S}.$$

Objective function:

$$\min \sum_{ij \in \mathcal{S} \times \mathcal{D}} c_{ij} f_{ij} + \sum_{i \in \mathcal{S}} C_i x_i.$$

Facility Expansion Planning Model

Defining the Subproblems

Supply constraints:

$$\sum_{j \in \mathcal{D}} f_{ij} \leq s_i + S_i x_i, \quad \forall i \in \mathcal{S},$$

Demand constraints:

$$\sum_{i \in \mathcal{S}} f_{ij} = d_j, \quad \forall j \in \mathcal{D},$$

Binary expansions:

$$x_i \in \mathcal{Z}_+ \quad \forall i \in \mathcal{S}.$$

Objective function:

$$\min \sum_{ij \in \mathcal{S} \times \mathcal{D}} c_{ij} f_{ij} + \sum_{i \in \mathcal{S}} C_i x_i.$$

Outline

Implementation using JuDGE

Formulating the subproblems

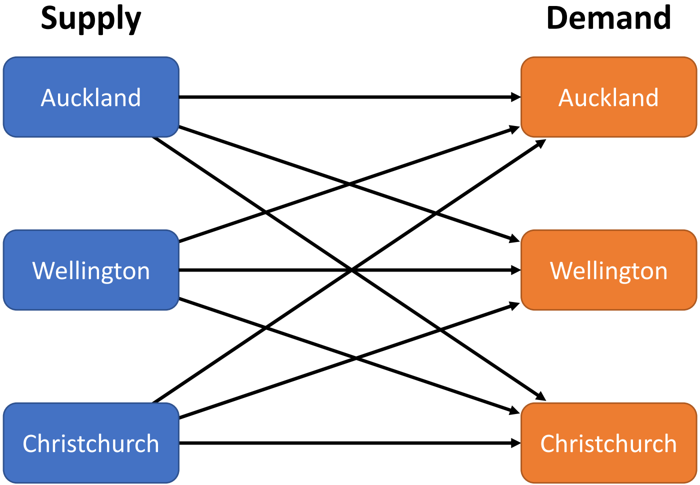
The subproblems are implemented as JuMP models, with additional macros, which are used to enable JuDGE to automatically formulate the master problem.

```
function sub_problems(node)
    model = Model(JuDGE_SP_Solver)

    @expansion(model, 0<=new_supply[supply_nodes]<=10, Int, lag=1)

    @capitalcosts(model, sum(C(node)[i]*x[i] for i in supply_nodes))

    @variable(model, x[supply_nodes, demand_nodes] >= 0)

    @objective(model, Min,
               sum(c[i, j] * x[i, j] for i in supply_nodes, j in demand_nodes))

    @constraint(model, Supply[i in supply_nodes],
                sum(x[i, j] for j in demand_nodes) <= s(node)[i] + S[i]*x[i])

    @constraint(model, Demand[j in demand_nodes],
                sum(x[i, j] for i in supply_nodes) == demand(node)[j])

    return model
end
```

Implementation using JuDGE

Formulating the subproblems

The subproblems are implemented as JuMP models, with additional macros, which are used to enable JuDGE to automatically formulate the master problem.



Implementation using JuDGE

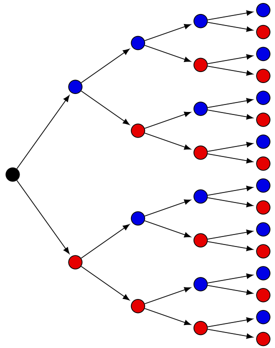
Creating a JuDGE Scenario Tree

There are several ways that trees can be created for JuDGE including:

- from a list of leaf nodes;
- from a file that specifies nodes, each node's parent and corresponding data;
- as a symmetric tree with constant depth and degree.

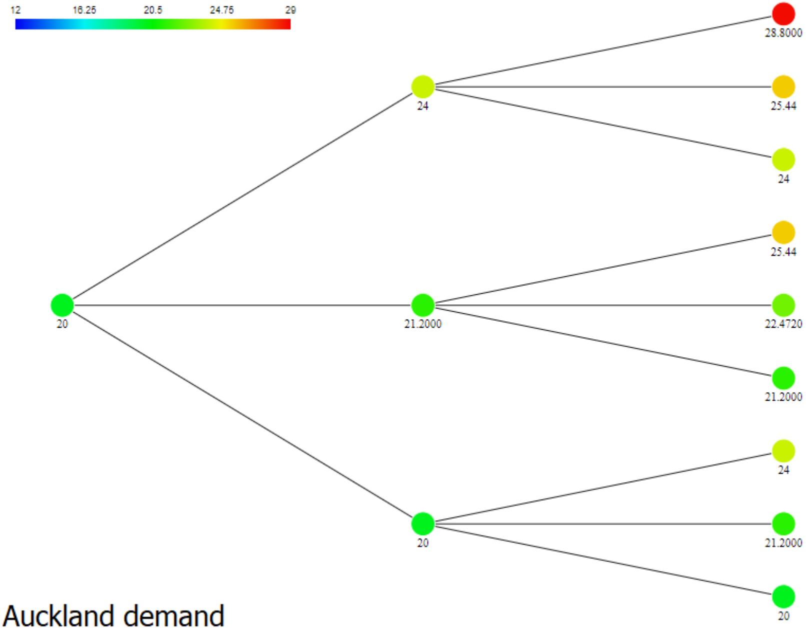
To create a tree with a depth 4 and degree 2:

```
tree = narytree(4,2)
```



Implementation using JuDGE

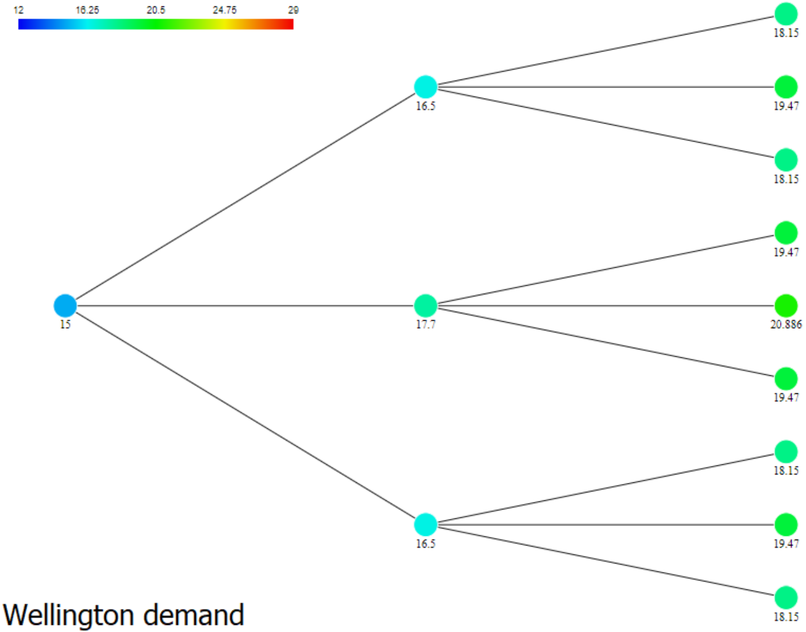
Creating a JuDGE Scenario Tree



Auckland demand

Implementation using JuDGE

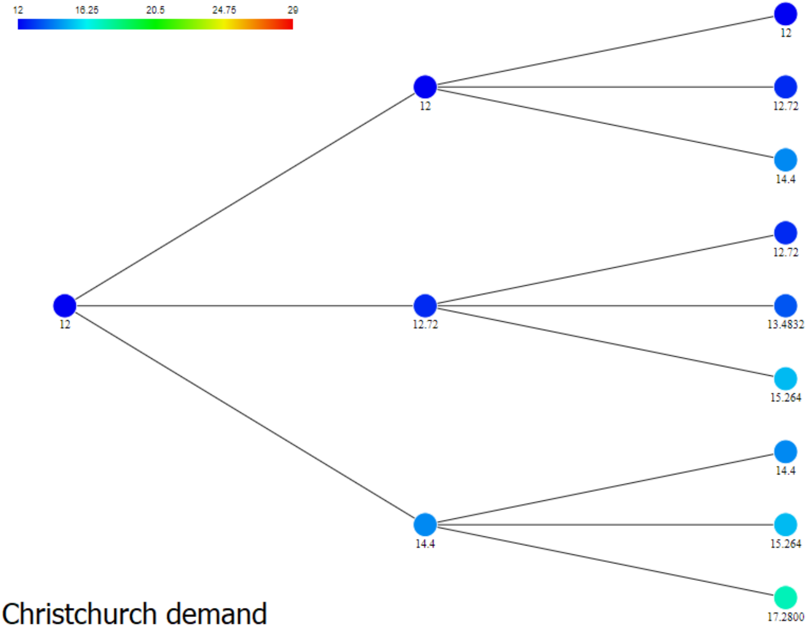
Creating a JuDGE Scenario Tree



Wellington demand

Implementation using JuDGE

Creating a JuDGE Scenario Tree



Christchurch demand

Implementation using JuDGE

Creating and solving the JuDGE Model

Once a tree has been created, and a function declared which defines the nodal subproblems, we can create a `JuDGEModel` as follows:

```
model = JuDGEModel(tree, ConditionallyUniformProbabilities,
        sub_problems,optimizer_with_attributes((method=GLPK.INTERIOR)
        -> GLPK.Optimizer(), "msg_lev" => 0, "mip_gap" => 0.0))
```

Implementation using JuDGE

Creating and solving the JuDGE Model

Once a tree has been created, and a function declared which defines the nodal subproblems, we can create a `JuDGEModel` as follows:

```
model = JuDGEModel(tree, ConditionallyUniformProbabilities,
        sub_problems,optimizer_with_attributes((method=GLPK.INTERIOR)
        -> GLPK.Optimizer(), "msg_lev" => 0, "mip_gap" => 0.0))
```

Implementation using JuDGE

Creating and solving the JuDGE Model

Once a tree has been created, and a function declared which defines the nodal subproblems, we can create a JuDGEModel as follows:

```
model = JuDGEModel(tree, ConditionallyUniformProbabilities,
        sub_problems,optimizer_with_attributes((method=GLPK.INTERIOR)
        -> GLPK.Optimizer(), "msg_lev" => 0, "mip_gap" => 0.0))
```

Implementation using JuDGE

Creating and solving the JuDGE Model

Once a tree has been created, and a function declared which defines the nodal subproblems, we can create a JuDGEModel as follows:

```
model = JuDGEModel(tree, ConditionallyUniformProbabilities,
        sub_problems,optimizer_with_attributes((method=GLPK.INTERIOR)
        -> GLPK.Optimizer(), "msg_lev" => 0, "mip_gap" => 0.0))
```

Implementation using JuDGE

Creating and solving the JuDGE Model

Once a tree has been created, and a function declared which defines the nodal subproblems, we can create a JuDGEModel as follows:

```
model = JuDGEModel(tree, ConditionallyUniformProbabilities,
        sub_problems,optimizer_with_attributes((method=GLPK.INTERIOR)
        -> GLPK.Optimizer(), "msg_lev" => 0, "mip_gap" => 0.0))
```

Implementation using JuDGE

Creating and solving the JuDGE Model

Once a tree has been created, and a function declared which defines the nodal subproblems, we can create a `JuDGEModel` as follows:

```
model = JuDGEModel(tree, ConditionallyUniformProbabilities,
        sub_problems,optimizer_with_attributes((method=GLPK.INTERIOR)
        -> GLPK.Optimizer(), "msg_lev" => 0, "mip_gap" => 0.0))
```

If the model passes the in-built testing, ensuring that the JuMP models are set up correctly, the model can be solved using the command:
`JuDGE.solve(model, Termination = termination(reltol=1e-4))`

There are several additional termination conditions that can be included as optional arguments: `rlx_abstol`; `abstol`; `rlx_reltol`; `reltol`; `time_limit`; `max_iter`.

Implementation using JuDGE

JuDGE Iterations

As JuDGE solves the problem, it reports the objective + the lower- & upper bound:

```
Termination            Absolute        Relative
---------------------------------------------
Binary/Integer:       1.0000e-10      1.0000e-10
Relaxation:           1.0000e-10      1.0000e-04
---------------------------------------------
Integer tolerance:    1.0000e-09
Time-limit:           Inf
Max iterations:       Inf
Allow fractional:     binary_solve
---------------------------------------------

Relaxed ObjVal |  Upper Bound   Lower Bound | Absolute Diff  Relative Diff | Fractional |     Time    Iter
           Inf |          Inf          -Inf |           Inf            NaN |          0 |    0.304       1
  1.253975e+06 | 1.253975e+06          -Inf |           Inf            NaN |          0 |    0.613       2
  1.939722e+05 | 1.939722e+05  6.460694e+04 |  1.293653e+05   2.002343e+00 |          0 |    1.123       3
  1.915015e+05 | 1.939722e+05  9.750023e+04 |  9.400132e+04   9.641138e-01 |          2 |    1.638       4
  1.914004e+05 | 1.914004e+05  1.240748e+05 |  6.732559e+04   5.426211e-01 |          0 |    2.223       5
  1.913624e+05 | 1.913624e+05  1.280194e+05 |  6.334291e+04   4.947914e-01 |          0 |    2.866       6
  1.906704e+05 | 1.913624e+05  1.815446e+05 |  9.125788e+03   5.026748e-02 |         12 |    3.457       7
  1.905671e+05 | 1.913624e+05  1.815446e+05 |  9.022535e+03   4.969873e-02 |          3 |    4.073       8
  1.905649e+05 | 1.913624e+05  1.815446e+05 |  9.020318e+03   4.968652e-02 |         12 |    4.687       9
  1.905557e+05 | 1.905557e+05  1.831433e+05 |  7.412418e+03   4.047332e-02 |          0 |    5.333      10
  1.905556e+05 | 1.905556e+05  1.899303e+05 |  6.253313e+02   3.292425e-03 |          0 |    5.911      11
  1.905425e+05 | 1.905556e+05  1.899303e+05 |  6.122061e+02   3.223320e-03 |          6 |    6.494      12
  1.905425e+05 | 1.905556e+05  1.903193e+05 |  2.231239e+02   1.172366e-03 |          6 |    7.096      13
  1.905423e+05 | 1.905423e+05  1.903193e+05 |  2.229904e+02   1.171665e-03 |          0 |    7.768      14
  1.905423e+05 | 1.905423e+05  1.903262e+05 |  2.160559e+02   1.135187e-03 |          0 |    8.335      15
  1.905423e+05 | 1.905423e+05  1.903263e+05 |  2.159938e+02   1.134860e-03 |          0 |    8.914      16
  1.905423e+05 | 1.905423e+05  1.903267e+05 |  2.155513e+02   1.132533e-03 |         20 |    9.487      17
  1.905423e+05 | 1.905423e+05  1.905423e+05 |  1.272894e-05   6.680375e-11 |         12 |   10.126      18
  1.905423e+05 | 1.905423e+05  1.905423e+05 |  1.208854e-05   6.344281e-11 |          0 |   10.149     19*
```

Outline

Computational Benchmarks

					0.1%			1.0%		
					JuDGE		DetEq	JuDGE		DetEq
Degree	Depth	Seed	Nodes	Variables	Gurobi	GLPK	Gurobi	Gurobi	GLPK	Gurobi
3	2	1	15	2778	4.8s	3.9s	5.4s	0.3s	2.1s	0.4s
3	2	2	15	2778	4.7s	4.0s	5.0s	3.5s	2.8s	0.4s
3	2	3	15	2778	2.2s	2.2s	5.1s	1.8s	1.9s	0.8s
3	2	4	15	2778	3.9s	3.8s	15.5s	2.8s	2.8s	1.7s
3	2	5	15	2778	2.1s	2.0s	3.3s	1.3s	1.6s	0.3s
3	3	1	40	8547	22s	28.90s	485.72s	10s	15s	80s
3	3	2	40	8547	28s	36.15s	1049.62s	17s	29s	122s
3	3	3	40	8547	37s	26.86s	53.20s	26s	23s	35s
3	3	4	40	8547	20s	23.77s	11.77s	7s	9s	2s
3	3	5	40	8547	23s	24.06s	0.36%	13s	17s	223s
3	4	1	85	18169	49s	41.44s	0.51%	21s	38s	116s
3	4	2	85	18169	82s	83.77s	0.74%	38s	45s	1855s
3	4	3	85	18169	102s	91.32s	1.75%	48s	42s	1.75%
3	4	4	85	18169	112s	89.37s	0.58%	40s	69s	845s
3	4	5	85	18169	54s	136.52s	0.29%	18s	22s	799s
4	4	1	341	72889	382s	565.38s	0.35%	86s	100s	758s
4	4	2	341	72889	250s	343.34s	0.32%	85s	94s	877s
4	4	3	341	72889	613s	656.49s	0.30%	116s	165s	572s
4	4	4	341	72889	327s	288.48s	0.34%	82s	104s	665s
4	4	5	341	72889	617s	459.89s	0.23%	83s	132s	577s
4	5	1	781	166978	1710s	1637.48s	0.96%	811s	1059s	5589s
4	5	2	781	166978	1906s	2280.90s	0.73%	337s	1165s	905s
4	5	3	781	166978	1429s	1614.45s	0.51%	353s	990s	5265s
4	5	4	781	166978	1756s	1637.31s	0.55%	559s	471s	3335s
4	5	5	781	166978	376s	273.93s	0.18%	376s	274s	641s
5	5	1	3906	835103	0.11%	2247.03s	0.94%	6358s	639s	2840s
5	5	2	3906	835103	0.15%	5407.42s	0.89%	3438s	753s	3080s
5	5	3	3906	835103	0.26%	3139.88s	0.90%	4379s	738s	4848s
5	5	4	3906	835103	6740s	2198.67s	0.39%	3616s	745s	3152s
5	5	5	3906	835103	0.16%	3049.85s	1.07%	3688s	753s	1.07%

Outline

Communication of JuDGE Solutions

Visualizing the policy

One of the challenges with stochastic multi-horizon optimization is the communication of an optimal policy.

Communication of JuDGE Solutions

Visualizing the policy

One of the challenges with stochastic multi-horizon optimization is the communication of an optimal policy.

JuDGE provides a custom framework to interactively explore the policy, enabling users to understand how the revelation of information influences the investment decisions, but also how these, in turn, affect the operational decisions in the short-term.

Communication of JuDGE Solutions

Visualizing the policy

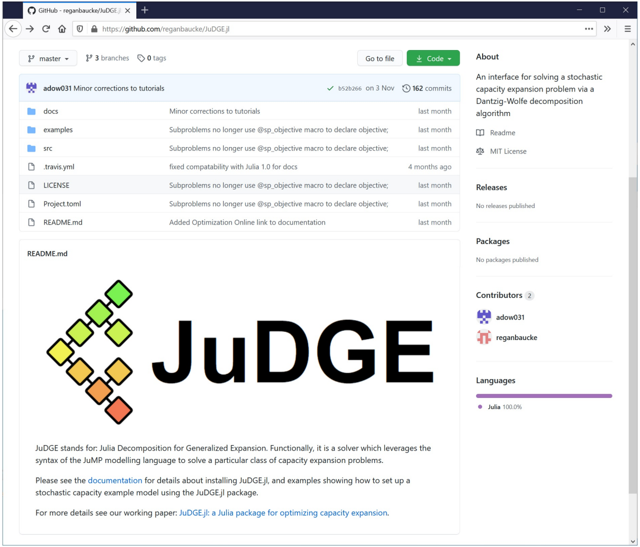
One of the challenges with stochastic multi-horizon optimization is the communication of an optimal policy.

JuDGE provides a custom framework to interactively explore the policy, enabling users to understand how the revelation of information influences the investment decisions, but also how these, in turn, affect the operational decisions in the short-term.

This framework is built around html and javascript, and therefore is very flexible, with the ability to integrate: maps, plots, svg graphics, or any other web-based visualization.

Installing and using JuDGE

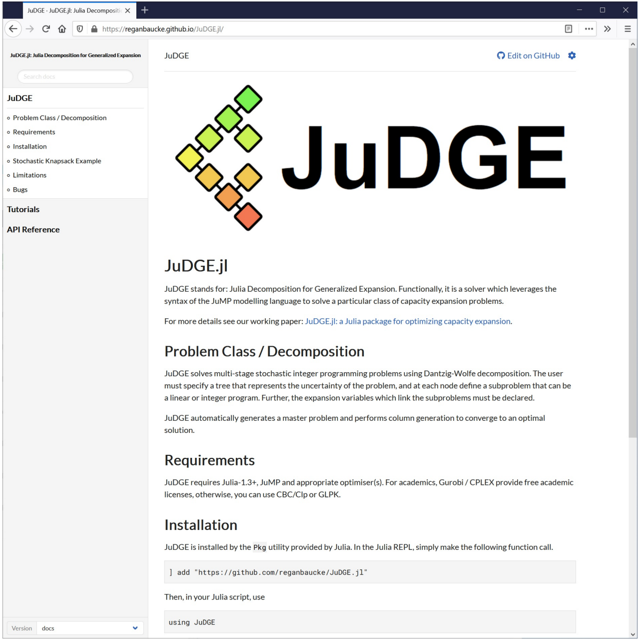
Github Repository



`https://github.com/reganbaucke/JuDGE.jl`

Installing and using JuDGE

Installing the JuDGE Package



JuDGE · JuDGE.jl: Julia Decompositi

https://reganbaucke.github.io/JuDGE.jl/

JuDGE.jl: Julia Decomposition for Generalized Expansion

Search docs

JuDGE

- Problem Class / Decomposition
- Requirements
- Installation
- Stochastic Knapsack Example
- Limitations
- Bugs

Tutorials

API Reference

Version docs

JuDGE

Edit on GitHub

JuDGE

JuDGE.jl

JuDGE stands for: Julia Decomposition for Generalized Expansion. Functionally, it is a solver which leverages the syntax of the JuMP modelling language to solve a particular class of capacity expansion problems.

For more details see our working paper: JuDGE.jl: a Julia package for optimizing capacity expansion.

Problem Class / Decomposition

JuDGE solves multi-stage stochastic integer programming problems using Dantzig-Wolfe decomposition. The user must specify a tree that represents the uncertainty of the problem, and at each node define a subproblem that can be a linear or integer program. Further, the expansion variables which link the subproblems must be declared.

JuDGE automatically generates a master problem and performs column generation to converge to an optimal solution.

Requirements

JuDGE requires Julia-1.3+, JuMP and appropriate optimiser(s). For academics, Gurobi / CPLEX provide free academic licenses, otherwise, you can use CBC/Clp or GLPK.

Installation

JuDGE is installed by the `Pkg` utility provided by Julia. In the Julia REPL, simply make the following function call.

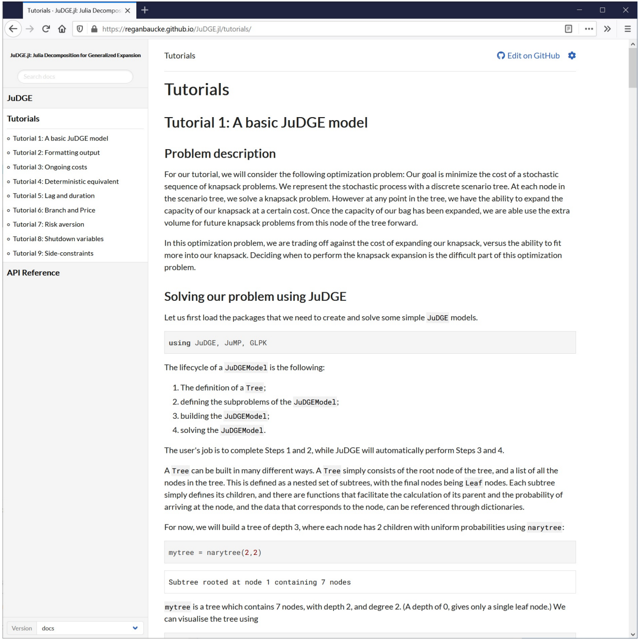
```
] add "https://github.com/reganbaucke/JuDGE.jl"
```

Then, in your Julia script, use

```
using JuDGE
```

Installing and using JuDGE

Tutorials and Examples



Thanks for your attention.

Any questions?

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

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