SINTEF

Automatic Decomposition of JuMP Models using TimeStructDecomposition.jl

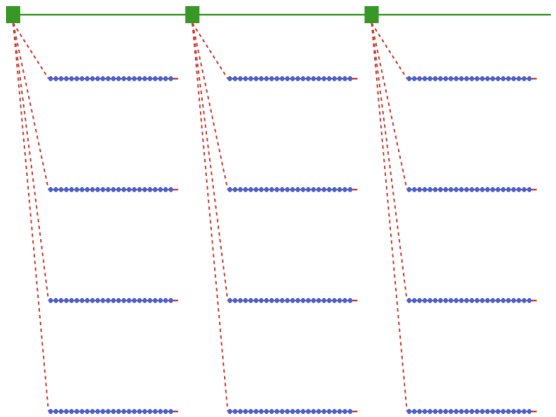
Truls Flatberg, Lars Hellemo, SINTEF

Motivation

- Multi-horizon optimization models can be too big to solve monolithically
- Using TimeStruct.jl you have access to the semantics of the time structure
- Write models with standard JuMP macros and decomposition is handled automatically
- Plug into general-purpose decomposition solvers
- Couple academic developments and practical use

TimeStruct.jl

- Package that supports the efficient development of optimization models with multi-horizon time modeling and possible uncertainty
- Provides a hierarchy of time structure types and the iteration/indexing machinery to go with them



```
# Two strategic periods, each with 8760 operational hours
periods = TwoLevel(2, 8760, SimpleTimes(8760, 1))

# With operational uncertainty: 4 scenarios per strategic period
periods = TwoLevel(3, 24, OperationalScenarios(4, SimpleTimes(24, 1)))
```

https://github.com/sintefore/TimeStruct.jl

https://www.youtube.com/watch?v=Hz6AL5kClKU

Structural key insight

Variable indices reveal the decomposition

```
using TimeStruct

periods = TwoLevel(10, SimpleTimes(24,1))
model = JuMP.Model()

@variable(model, invest[strat_periods(periods)], Bin)
@variable(model, charge[t in periods] ≥ 0)
@constraint(model, soc[t] ≤ capacity[sp])
```

master problem

one subproblem per strategic period

linking constraint

Usage

Write your model with normal JuMP macros without structural changes

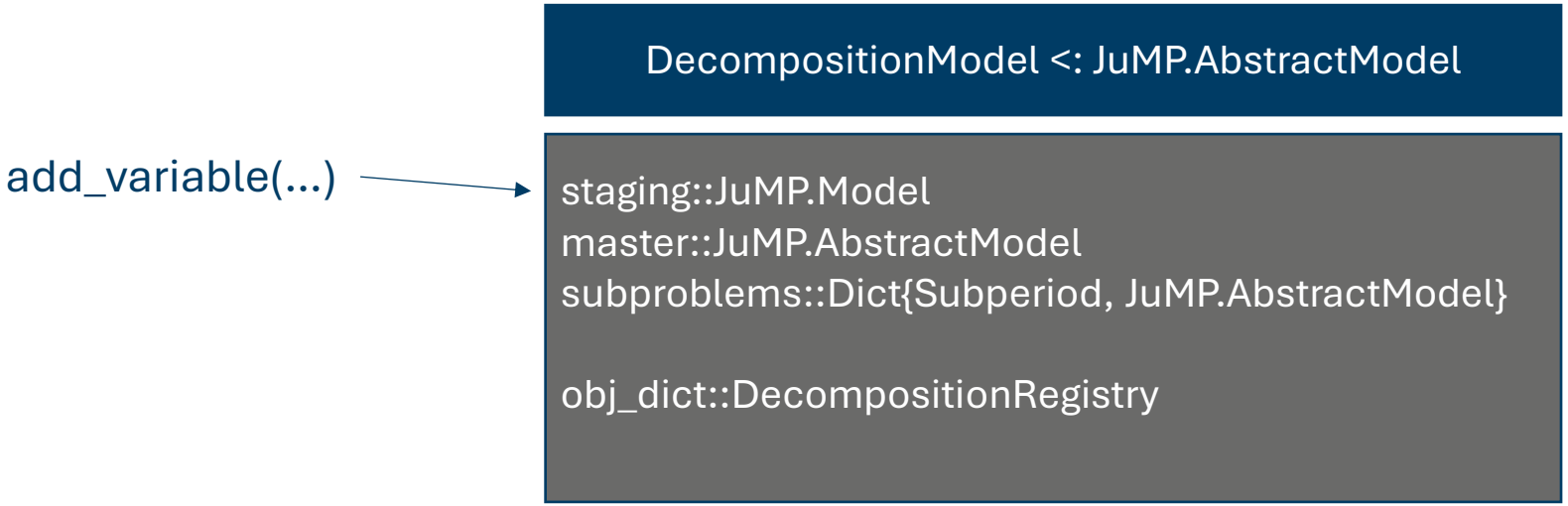
```
using TimeStruct, TimeStructDecomposition

periods = TwoLevel(10, SimpleTimes(24,1))
model = DecompositionModel(periods, StrategicDecomposition())

@variable(model, invest[strat_periods(periods)], Bin)
@variable(model, charge[t in periods] ≥ 0)
@constraint(model, soc[t] ≤ capacity[sp])
```

`DecompositionModel` automatically classifies variables and constraints to a master model and per-strategic-period (or per-scenario) sub-models at declaration time, based on the index type of each variable container.

How does it work internally

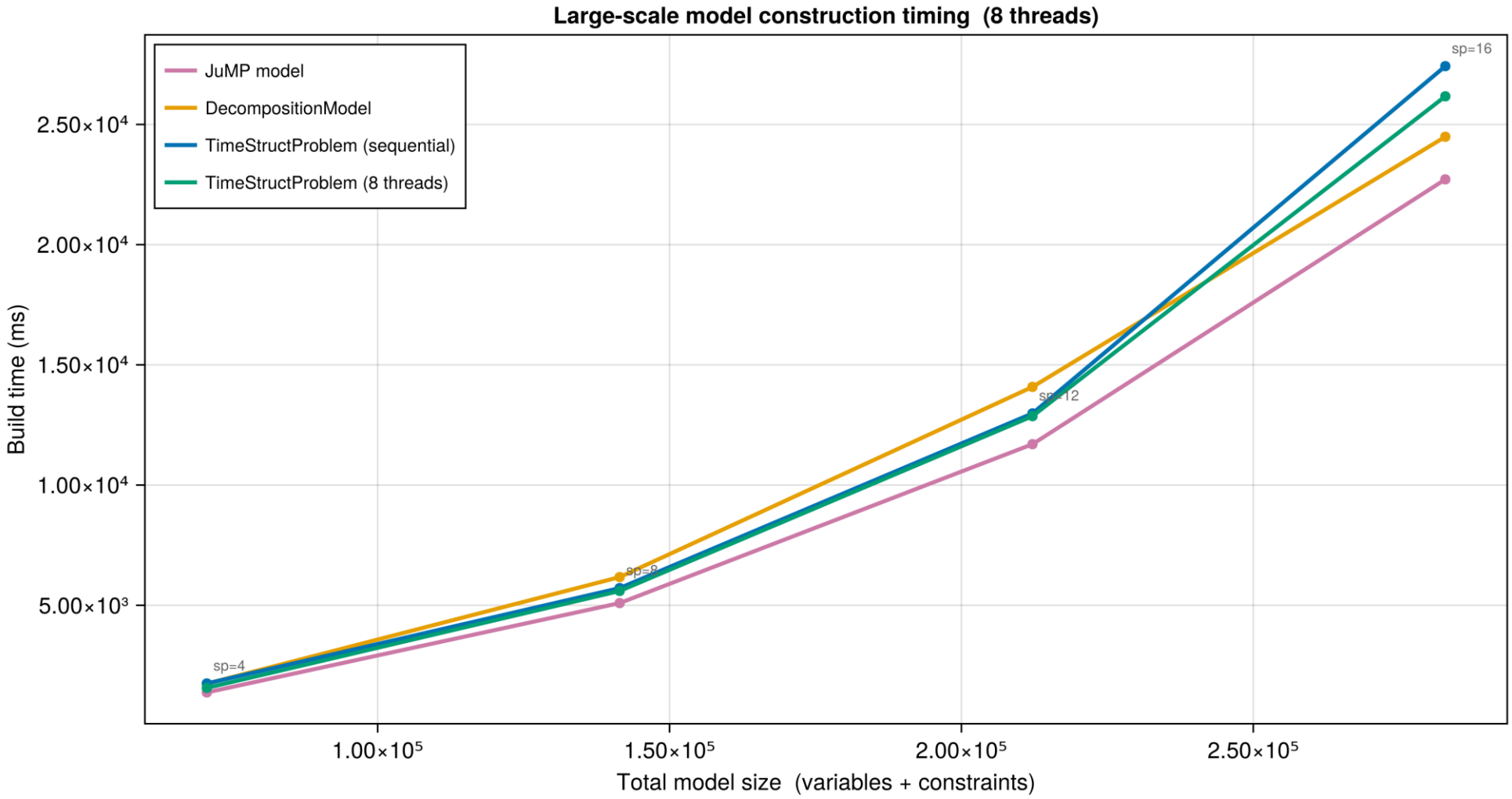


When a new variable is registered in the model dictionary, the `setindex!` of the dictionary triggers a classification of the variable container (master/subproblem) by checking indices

```
model[:name] = var
```

Supported containers: `DenseAxisArray`, `SparseAxisArray`, `IndexedVarArray`

Computational overhead



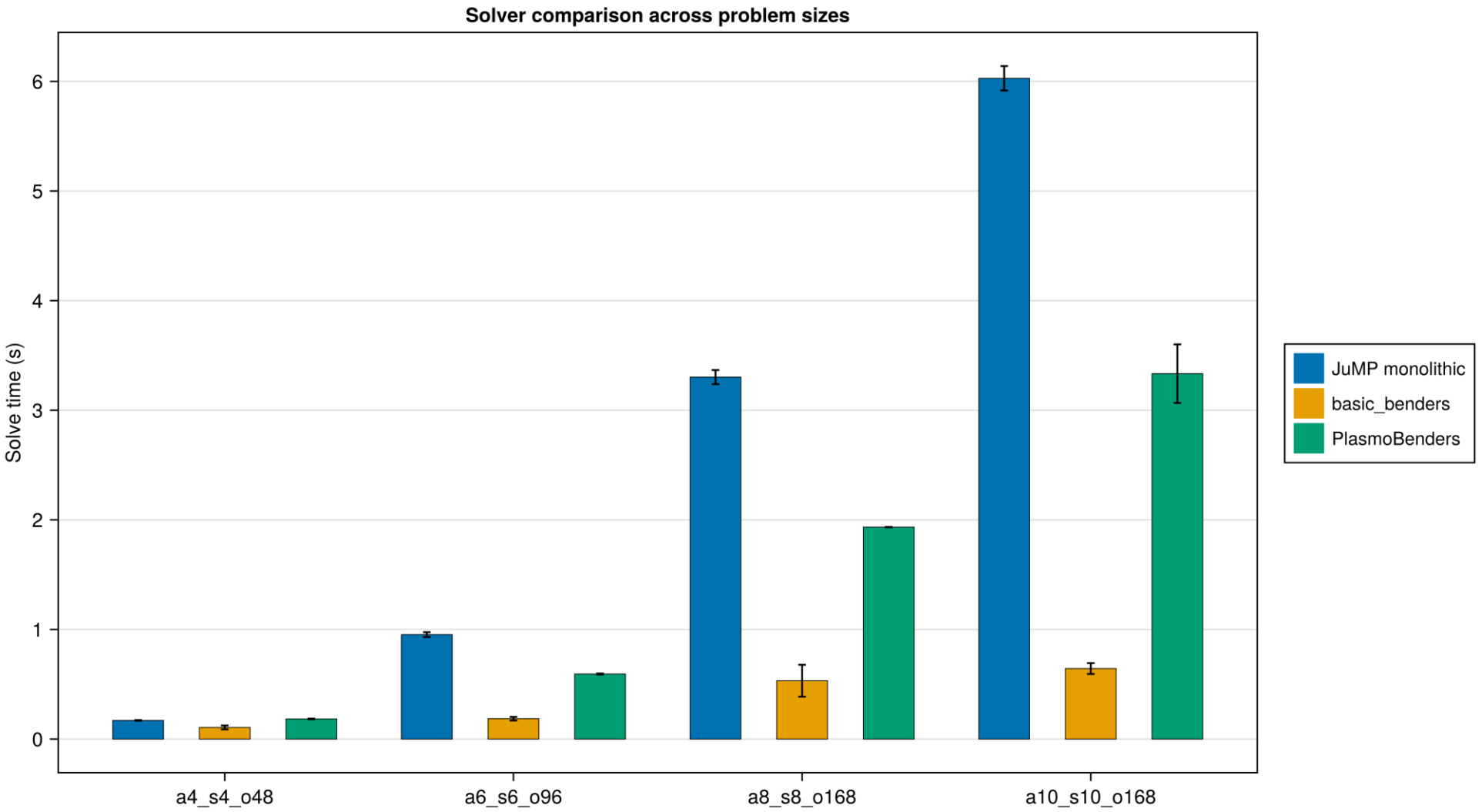
Solving

- The decomposed problem can be solved using different backends
  - Custom Benders' decomposition (multithreaded and distributed support)
  - Benders' using `PlasmoAlgorithms.jl`

```
optimize_with_decomposition!(model, periods, PlasmoBendersAlgorithm())
```

Performance

solver = HiGHS, nthreads = 4



Final remarks & takeaways

- Automatic decomposition with zero model refactoring
- Scalable solution approach for large multi-horizon models
- Flexible solver integration
- Validated on real-world energy models
  - EnergyModelsX
  - OpenEMPIRE
- Developed within the InterPlay Research Centre

Technology for a better society

SINTEF.NO