This is a demonstration of a Transmission Expansion Planning (TEP), a way of planning the building of an electric infrastructure system. In this TEP presented here, a central planner decides where to construct new lines based on the goal of maximizing social welfare and minimizing construction costs, while generation is controlled by private companies motivated by profitability. Social welfare in this scenario is defined as meeting all electrical demand of the population consistently and as economically as possible, and costs include both yearly line construction costs and hourly generation outlays on generation. While research into efficient solutions of the deterministic version of TEP have been ongoing for decades, introducing probabilistic constraints, such by attempting to estimate demand decades into the future, can greatly increase the complexity and create intractable models even on the most modern computer hardware. One way to balance the conflicting demands of needing a model solvable in a reasonable amount of time, while also adequately addressing the extra complexity of parameter uncertainty is to use robust optimization. However, robust optimization assumes that some, or all, uncertain parameters take a worse-case or nearly worse case realization, but such assumptions run the risk of over-conservative decisions [1] So far, all programs assume a DC simplification of the AC power flow through the netwrok. This is necessitated by the fact that modeling power as AC creates a highly nonconvex problem that can not be solved with linear solvers. See [4] for a discussion of the differences between various models of transmission.

Implementation will be on Pyomo (www.pyomo.org/), an optimization modeling language written in Python [2][3]. Since Pyomo is an open source software, and most other work on robust TEPs have used algebraic modelling software that require users to purchase a license, there is hopefully demand for this development. If not, well it's here anyway ¯_(ツ)_/¯. Resources about Pyomo that may be helpful:

Basic Documentation: https://pyomo.readthedocs.io/en/stable/

Git Repository: https://github.com/Pyomo

Installation

To run a user needs to have Python and Pyomo installed (www.pyomo.org/). Pyomo works by calling an external Mixed Integer Linear Programming (MILP) solver which also must be separately installed. Experimental results provided in this repository are found using the CPLEX solver from IBM (www.cplex.com). Another option is the Gourbi solver(www.gurobi.com/). However, these both require the purchase of a license if you are a non-academic user. An option with a free software license is the GLPK solver from the GNU project (www.gnu.org/software/glpk/) However, this list is not exhaustive, so reference the Pyomo documentation for all possibilities.

Note this software is still very much in development and should not be used by anyone, living or dead, for any purpose commercial, academic or personal.

Tests

These provide a series of tests to establish the validity of the different aspects of the program by comparing to known solves or to solves by a different optimization software programs. Note, that test models may use different setups of parameters to match difference in models in the literature See:

github.com/Homersmyid/TEP/blob/master/Tests/Readme.pdf

for a description of the tests. Each provides a self-contained code with accompanying data file. Also provided is the parameter and solutions if possible.

Files

RuizMain.py

Implements the column-and-constraint method from Ruiz and Conjego [5]. Additional information is provided by [1], in the chapter on TEP problems. As of now, a constant denotes the initial infrastructure setup of the model. See the file MainTheory for the theoretical description of the problem.

Input – None

Output – **To Screen.** Output the upper bound and lower bound for each iteration, as well as the absolute and relative gap between the two. Also, outputs the Pyomo.out.solve() for each run of the master and subproblem (see Pyomo documentation for more details.)

Constants

Start_x_star – A list of possible connections between nodes. Each tuple contains three numbers representing the starting node, ending node, and number of lines on connection. 0 represents no connection.

Stop – Number of iterations to stop after. All examples in the literature have finished in 10 iterations or less.

RuizC.py

The file RuizC.py contains constants to use for the program.

Input / Output - None

Constants

Solver – Pyomo calls an external solver to solve a MILP. As such, the user can decide any solver they have installed. Experimental results are found using the CPLEX solver from IBM (www.cplex.com). However, to use this requires to purchase a license if you are a non-academic user. An option with a free software license is the GLPK solver from the GNU project (www.gnu.org/software/glpk/)

Epsilon – The relative distance between the upper bound and lower bound found for the optimal solution to stop the algorithm at:

$$(Upper - Lower)/Upper$$

So, for example with an epsilon of $1e^{-6}$ and upper bound of $2e^{-8}$, the algorithm will stop if the lower bound is found to be within 200. A standard is $1e^{-6}$.

Data – The location of the data file. Assumed to be an AMPL style data file. See the documentation of Data Files to see how to set up the file.

Unctol – A tolerance to use on deciding if the data file has the same upper bounds as lower bounds on any set of uncertain parameters. This is to avoid a divide by zero situation.

MIPGAP – If using the CPLEX solver, the relative gap to stop the mixed integer solver at. See CPLEX documentation for more information

RuizSub.py

This implements an abstract model in Pyomo of the column-and-constraint method's subproblem portion. The subproblem combines the two-level non-linear problem of $\max_{d \in D} \left(\min_{y \in \Omega(x,d)} b^t y \right)$ into a single level linear problem. In application, this represents producers both minimizing their hourly costs and meeting demand for the worst-case possible realization of uncertain parameters. The method accomplishes this by linearizing with a "Big M" method and applying the KKT conditions to replace the innermost minimization. The scheme also includes an uncertainty budget to allow a modeler to avoid overly conservative estimates. See file SubTheory for a full description of the theory behind this model or a discussion of setting the uncertainty budgets.

Input – **From user.** None
From main. X_star. The set of built lines from the subproblem.
From data file. AMPL style data for a concrete model. See docs/Input.pdf for more.

Output – **To main.** Results of a Pyomo.opt.solve. (See Pyomo documents for details)
Value of maximization solve as a possible upper bound. The new objective value is only saved by Main if it less than previous upper bound.

RuizMast.py

Implements the master problem of the column-and-constraint method. Has two components, the abstract model in Pyomo of the master problem and a function for adding variables and constraints to the concrete model on every new iteration. Please see file MastTheory for a full description of the theory behind setup.

Abstract Model

This method takes in the realizations of uncertain parameters, both supply and demand, from previous subproblems as constant parameters. Then, it outputs the choice of lines that will minimize both line construction costs and hourly costs, as well a new lower bound, which will always be less than or equal to previous lower bound.

Input – **From user.** None
From main. X_star. A list of already existing power lines.
From data file. Data for concrete model. See docs/Input.pdf.

Output – **To main.** Results of a Pyomo.opt.solve. (See Pyomo documents for details)
Value of maximization solve as a lower bound.

Function

This function adds a new series of constraints based on the generation and supply level found from the last subproblem solve:
mast_func(imast, subdem, subgenpos, in_x_star, k)

Input - Imast. The concrete version of the master problem.
Subdem. Demand at each sink. From last subproblem solve.
Subgenpos. Generation capacity at each generator. From last subproblem solve.
In_x_star. A list of preexisting power lines.
K. What iteration the main is currently on.

Output – Makes changes in imast function.

References

[1] Antonio J. Conejo, Luis Baringo Morales, S. Jalal Kazempour, and Afzal S. Siddiqui. 2016. Investment in Electricity Generation and Transmission: Decision Making Under Uncertainty (1st ed.). Springer Publishing Company, Incorporated.

[2] Hart, William E., Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola. *Pyomo – Optimization Modeling in Python*. Second Edition. Vol. 67. Springer, 2017.

[3] Hart, William E, and David L Woodruff. "Pyomo: Modeling and Solving Mathematical Programs in Python." *Mathematical Programming Computation*, vol. 3, no. 3, 2011, pp. 219–260.

[4] Romero, R., A. Monticelli, Ae Garcia, and S. Haffner. "Test systems and mathematical models for transmission network expansion planning." IEE Proceedings-Generation, Transmission and Distribution 149, no. 1, 2002, 27-36.

[5] Ruiz, C, and A.J Conejo. "Robust Transmission Expansion Planning." *European Journal of Operational Research*, vol. 242, no. 2, 2015, pp. 390–401.