Minpower: A Power Systems Optimization Toolkit

Adam Greenhall and Rich Christie
Dept. of Electrical Engineering
University of Washington
Email: {argreen,richc}@uw.edu

Jean-Paul Watson
Discrete Math and Complex Systems Dept.
Sandia National Laboratories
Email: jwatson@sandia.gov

***Abstract*—Minpower is an open source toolkit for students and researchers in power systems optimization. The toolkit is designed to make working with the classical problems of Economic Dispatch (ED), Optimal Power Flow (OPF), and Unit Commitment (UC) simple and intuitive. Minpower is also built for flexibility and will be a platform for research on smart grids and stochastic resources. Powerful generic optimization solvers (e.g., CPLEX) can be used by Minpower, allowing for reasonable solution times on even large-scale problems. The goal is to create a state-of-the-art open source tool that enables collaboration and accelerates research and learning. The Minpower code is open source and is set up for collaborative authorship and maintenance. Tutorials and thorough documentation are available at minpowertoolkit.com.**

***Index Terms*—Power systems, Optimization, Power generation economics, Power system analysis computing, Power engineering education.**

I. Introduction

The electrical power system is being re-imagined; more complex and less predictable technologies are beginning to replace the well-known, deterministically controlled technologies we rely on today. At the same time, electricity production will increasingly become governed by markets which strive to lower overall system costs. The next generation of engineers working on the power system will see high penetrations of renewables and demand that responds to the price of energy. These engineers will work across a number of disciplines – time-varying statistics, optimization, economics, software development, and high performance computing. They will need tools to help them.

Minpower is an open source toolkit for power systems optimization. It currently considers the three classic power systems operational problems:

- ED Economic Dispatch allocates available generation to meet the current system load at minimal cost.
- OPF Optimal Power Flow allocates available generation to meet the current load at each electrical bus while minding transmission system limits. OPF incorporates the electrical system topology into the ED problem.
- UC Unit Commitment schedules generation (on or off) to satisfy the predicted load of the next day(s). UC adds a dimension of time to the ED problem.

These classical problems are in a state of flux due to the introduction of markets and renewables into the power system. These changes have made the problems more complicated to formulate and more difficult to solve. Faced with these challenges, developers often trade off a more complex formulation for faster solution times or vice versa. Minpower is designed to be flexible, extensible, and fast enough for researchers and developers working on the evolution of these problems, while remaining easy to use for students learning the classical formulations.

This paper highlights the features that distinguish Minpower and describes the existing state-of-the-art tools Minpower is built upon. Demonstrations are given for two textbook problems in §VI-A,§VI-B. Performance on a research-scale test case is discussed in §VI-C. The addition of a more complicated UC feature – the ability to shed load – is discussed in §VI-D.

II. Existing Tools

A. Power Systems

There are many power systems operations tools available, but few are open source and none provide a flexible, powerful, and simple interface to solve unit commitment problems. The US power system currently depends on commercial software for economic optimization, supplied by vendors like Alstom Grid, GE, Siemens, and ETAP. However, these tools are too expensive and complex to be effectively used in research or teaching. There are several open source power systems tools for power flow, OPF, and dynamic analysis[1], including MatPower[2], PyPower[3], AMES[4], PSAT[5], MatDyn[6], and InterPSS[7]. However, these tools are mainly aimed at teaching and solving small-scale problems and none of them includes the capability to solve a UC. Many of these tools also depend on MatLab, a commercial platform which has advantages in numerical computation but compares poorly with higher-level open source languages like Python in areas like object oriented modeling[8]. In addition, Python can easily call efficient external libraries for performing time-consuming computations[9]; this approach is used for optimization in Minpower.

B. Optimization

There are many optimization modeling systems that allow for a reasonably high-level specification of a problem and subsequent solution by a modern mixed integer linear

programming solver (e.g., CPLEX). Commercial optimization software packages such as GAMS, AMPL, and AIMMS are often found in industry, but require expensive licenses to be able to solve large problems. Open source alternatives include GLPK[10], the COIN optimization research suite[12], and Coopr[13]. GLPK is a free, open source package that includes the MathProg modeling language and the GLPSOL solver. However, GLPK is currently not suitable for research due to slow performance when solving large problems. The COIN suite includes CMPL, a mathematical programming language which has the ability to be used with other solvers, but requires learning a syntax unique to CMPL. Minpower uses Coopr, which is described in §V. However, even when using one of these higher level modeling systems, creating a UC problem and understanding the solver's results is a difficult programming task and requires considerable knowledge of mathematical optimization.

C. Minpower

Minpower was designed to fill the gap between power systems and optimization with the best tools available. By relying on open source software, Minpower avoids reinventing the wheel, harnesses expert knowledge from other domains, and remains freely available for student use. By using an open source solver (e.g., GLPK, SCIP[11], or COIN's CBC) or a solver with a free academic license (e.g., CPLEX or Gurobi) in combination with Minpower, students can obtain a full power systems optimization platform at no cost. Researchers can continue to use the powerful optimization solver of their choice in combination with Minpower's easy to use, extensible tools.

III. Minpower Features

A. Unit Commitment

UC is the most actively researched of the three classical problems and is also the most complex and least covered by existing tools. Minpower includes a full-featured, integrated method for UC formulation, solution, and graphing. A key feature – demonstrated in §VI-D – is the ability to easily add constraints or modify system modes (e.g., wind shedding or load shedding).

a) UC Formulation: The constraints implemented in the Minpower UC currently include: generator power limits, ramp rate limits, and up/down time limits. The inter-temporal constraints follow the formulation of Carrión and Arroyo[14]. Fuel costs can be represented by an arbitrary polynomial (and are converted to a piece-wise linear formulation automatically if the polynomial is non-affine). Simple start up and shut down costs can also be specified.

b) UC Intervals: Minpower can handle arbitrary commitment intervals because it interprets dates and times and includes them in the model. The default interval is one hour, but other intervals, such as 5min or 2hr, can be used simply by creating a load profile spreadsheet with longer or shorter intervals (e.g., the schedule in Table VI).

c) Rolling UCs: Simulating several days to a year of operation is now a common research task due to the variability of wind energy production. This simulation is usually done by running sequential commitments, with the final condition of one commitment becoming the initial condition for the next. Often the commitments are set up to overlap – e.g., run a 36 h long commitment every 24 h – to avoid the horizon effect. This effect occurs when generators are left in an ending state from which it will be expensive to meet the next commitment's first few hours of load. This is a relatively complex procedure that can be done automatically with Minpower.

B. Economic Dispatch

Creating and solving an economic dispatch problem with Minpower mirrors the formulation and features found in the UC, without the inter-temporal constraints and binary commitment variables.

C. Optimal Power Flow

Currently Minpower implements an active power only OPF, called DC OPF or Linear Programming (LP) OPF. This formulation may be inaccurate for a power systems experiencing reactive power issues. However, the DC OPF formulation is increasingly being used in day-ahead electricity markets because it can be solved by standard MIP solvers capable of handling large problem sizes. The objective of including OPF in Minpower is to provide a simple teaching tool and an architecture which can be extended for research on UC problems with transmission system constraints.

IV. Minpower Usage

Minpower is designed to have both a fast learning curve for students and enough capability and flexibility for significant research. The basic steps to operation Minpower are:

1) Specify the problem in spreadsheet form
2) Run the minpower script from the command line
3) Examine the results in spreadsheet and graphical form

None of these steps requires more than a basic understanding of any of the tools Minpower uses internally. Simply by setting a script parameter, Minpower can be directed to use one of many available solvers, including CPLEX, Gurboi, GLPK, and the full list supported by Coopr[15].

Researchers and developers who want more options or only specific components to be run can interact with the well-documented Application Programming Interface (API). Researchers can also write and integrate their own power system component models (e.g., demand response or energy storage).

V. Implementation

Minpower is written in Python, a high-level, readable programming language with an active development community. Minpower relies on several other well-developed

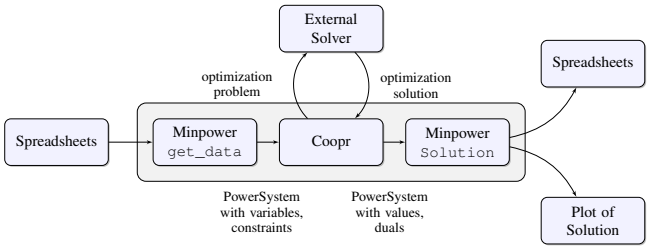

Fig. 1. Minpower process flow

open source Python packages to do the "heavy lifting" in areas such as optimization and visualization. Coopr is an optimization package which can use any major solver (e.g., CPLEX, Gurobi, GLPK)[15]. Coopr provides an optimization modeling language, called Pyomo, and a stochastic modeling and solver engine, called PySP[13, 16]. Matplotlib is a data visualization package styled after MatLab and developed by the Python community[17]. SciPy is a scientific computing package for Python that includes tools for matrix and polynomial mathematics[9].

Minpower has an object oriented design. The power system components – generators, loads, buses, and lines – are the core of this design. The hierarchy of these components is similar to the design proposed by Selvan[18]. However, in Minpower each power system component is treated as an optimization object and has variables, constraints, and – where appropriate – a contribution to the objective function (e.g., generator fuel cost).

When Minpower is called from the command line, the data is read from the spreadsheets in Comma Separated Value (CSV) format and used to create the power system objects. These generator, bus, load, and other objects are then used to construct the optimization problem using the Coopr package. The problem then gets passed off to an external solver, which can be chosen by passing the solver name in an argument to the Minpower script. The solver returns an answer (solution status, objective function, variable values, constraint duals, etc.) which is parsed and returned to the optimization objects by Coopr. Minpower then compiles the relevant results and uses them to make a spreadsheet and a plot of the solution using Matplotlib.

A. Documentation

A critical part of any piece of software is the documentation. Minpower's documentation is online at minpowertoolkit.com along with tutorials for ED, OPF, and UC problems. The documentation is built using Sphinx[19] and is updated with each new release.

B. Testing

A test suite is an essential tool for development - without testing changes cannot be made to the code with any certainty that they are working as intended and do not break other pieces of the code. The Minpower toolkit comes with a set of example problems defined in spreadsheets (that serve as an integration test suite) and a set of simple tests for individual constraints using the API (that serve as a unit test suite).

TABLE I
GENERATOR PARAMETERS FOR ED EXAMPLE

heat rate equation	P min	P max	fuel cost
`225+8.4P+0.0025P^2`	45.0	450	0.80
`729+6.3P+0.0081P^2`	45.0	350	1.02
`400+7.5P+0.0025P^2`	47.5	450	0.90

TABLE II
LOAD PARAMETERS FOR ED EXAMPLE

name	power
load	500

C. Collaboration Framework

The collaboration framework is also critical for open source software. Minpower uses GitHub for collaboration, code hosting, and bug tracking. The ethos of GitHub is that a project can be copied and worked on ("forked") by anyone. The fork can be easily "merged" back into the original project (ease of merging is a main feature of the Git revision system) or take on a life of its own. In this way, code can be collaborated on by many many authors and can continue to evolve after its originators move on.

VI. DEMONSTRATIONS

A. Economic Dispatch: a Textbook Example

The parameters associated with a textbook ED problem [20, prob.3.7] are described in Tables I, II. These tables are exact representations of the input spreadsheets used by Minpower. The three generators have quadratic heat rate curves (in MBtu/MWh), fuel costs (in $/MBtu), and power limits (in MW). The real power demand is in MW. In general, the units for Minpower input parameters are fixed, as described in the documentation[21].

The results returned by Minpower are shown in Table III and Fig.2 (with power in MW and incremental cost in $/MW). Equal Incremental Costs (ICs) are expected in a linear ED solution, but in this problem the cost curves are quadratic. The power values from the solution of the linearized problem are used to calculate the ICs on the original curves, resulting in small differences.

TABLE III
ED SOLUTION – GENERATOR POWERS AND INCREMENTAL COSTS (ICS)

Power	IC
216.0	7.58
75.5	7.67
208.5	7.69

TABLE IV
GENERATOR PARAMETERS FOR TEXTBOOK UC EXAMPLE

name	Pmin	Pmax	heat rate equation	fuel cost	start up cost	min up time	min down time
g1	50	200	220+9.9P	1.4	560	8	8
g2	15	60	80+10.1P	1.4	210	8	8
g3	15	50	60+10.8P	1.4	147	4	4
g4	5	40	40+11.9P	1.4	0	4	4
g5	5	25	34+12.14P	1.4	0	4	4

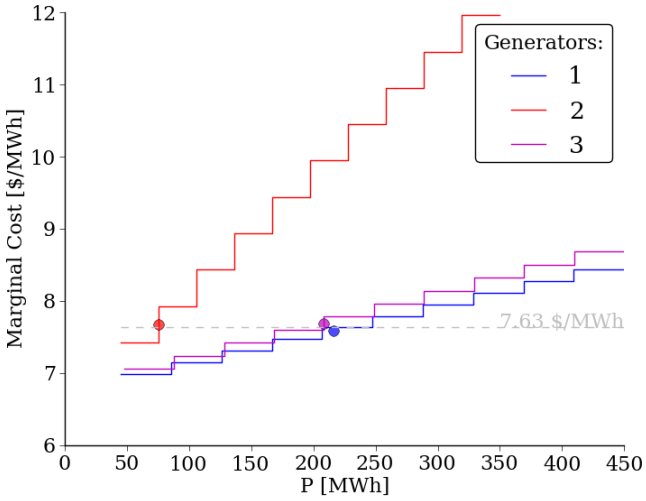

Fig. 2. ED solution showing linearized ICs for generators. Ten linear segments are used to model each generator's cost curve. Each of these linear segments translates to a constant segment in the marginal cost space.

TABLE V
GENERATOR INITIAL STATUS FOR UC EXAMPLE

name	power	status	hours in status
g1	250	1	4
g2	0	0	8
g3	0	0	8
g4	0	0	8
g5	0	0	8

B. Unit Commitment: a Textbook Example

Tables IV, V, VI describe the parameters for a UC problem, based on [20, prob.5.2]. Fig.3 shows the result of this commitment in stacked bar form. Minpower created, solved, and saved results for this problem in less than one second using the Gurobi solver on a desktop computer. This example illustrates Minpower's handling of time intervals. By default Minpower uses a commitment interval equal to the time intervals in the load profile, in this case 2 h. Note that this textbook case works well for testing constraints, but does not follow the typical patterns of a real power system (i.e., slowly varying load).

TABLE VI
LOAD PROFILE FOR UC EXAMPLE

time	power
0:00	250
2:00	320
4:00	110
6:00	75

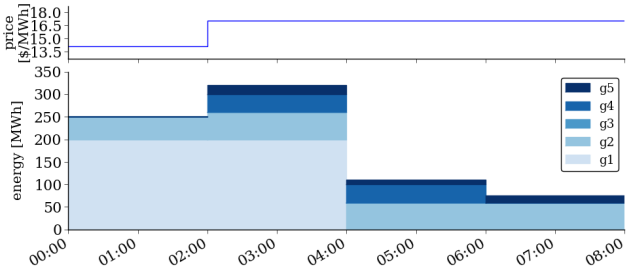

Fig. 3. UC solution for textbook example. The generation committed to meet the load in each two hour time period is shown in stacked bar form on the lower plot. The system price (i.e., the dual of the power balance constraint in the LP relaxation) is shown above.

TABLE VII
FUEL PRICES FOR ERCOT SIMULATION IN $/MBTU

fuel type	price
uranium	0.65
natural gas	4.00
coal	2.00

C. Unit Commitment: a Large-Scale Test Case

A research scale test case was developed to model the Electric Reliability Council Of Texas (ERCOT) Independent System Operator (ISO). The model includes 251 thermal generating units. Wind farms and Combined Heat and Power (CHP) plants are aggregated and taken as non-controllable units. Hourly wind and load data are available upon request from ERCOT. Generator information comes from the EPA's eGrid 2010 dataset[22]. Hourly energy and heat data for emissions-producing thermal units is available from the EPA's Clean Air Markets program[23]. This hourly data was analyzed to set heat rates, power limits, ramping limits, and up/down time limits for each thermal unit in the model. Nuclear units are modeled according to parameters taken from [24]. Start up costs are based on [25]. Fuel prices were assumed to be constant at the rates shown in Table VII. A rolling unit commitment (with hourly intervals and a 36 h

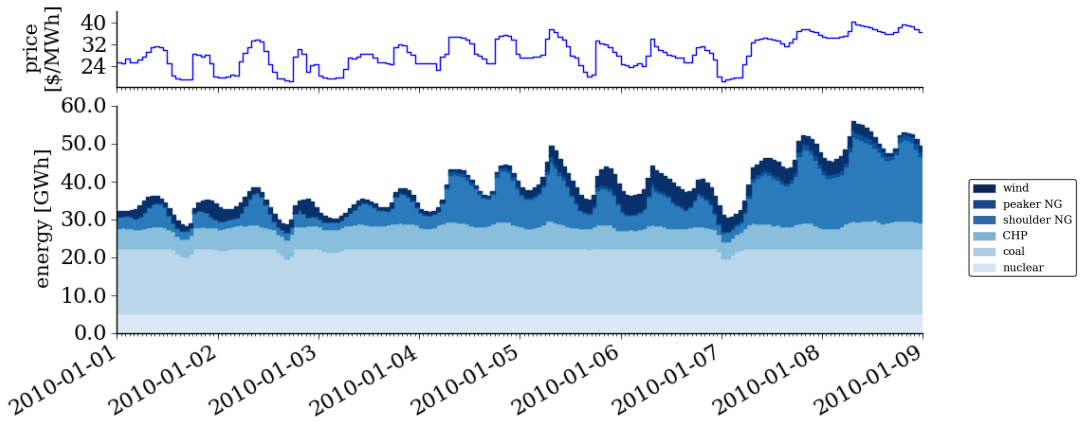

Fig. 4. UC results for the ERCOT simulation over eight days. An overlapping, rolling commitment was run with one 36 h UC solved for every 24 h period. In the figure, generation is grouped by unit type and shown in stacked bar form. The system price is shown above.

length UC run for every 24 h period) was run for the full 2010 year. For this simulation, a single electrical bus model was used. Fig.4 shows a portion of the results for the first eight days of January 2010. The problem was solved by a PC running Linux with 12 GB of RAM, four 2.67 GHz processors, and the Gurobi solver. The total Minpower run-time for the year was 9 h 47 min. The total cost of the solution for the year was $11.467 billion.

D. Augmenting UC with Load Shedding

A key feature of Minpower is the ability to easily modify the optimization problem. This is enabled by Minpower's structure for optimization and Python's simple syntax. To illustrate this feature, this section describes the steps involved in adding simulated load shedding – the emergency cutting of power to portions of the load when generation cannot meet demand. The Minpower code for the `Load` class (without shedding) is shown as follows:

Listing 1. Load class with no shedding

```
class Load(OptimizationObject):
    def __init__(self,name,bus,schedule):
        #load in inputs and set up
            optimization structure
        update_attributes(self,locals())
        self.init_optimization()
    def power(self,time): return self.
        schedule.get_energy(time)
    def create_variables(self,times):
        pass #no variables to create
    def create_objective(self,times):
        return 0
```

Like all of Minpower's power systems components, the `Load` class inherits structure and functionality from the `OptimizationObject` class. Without load shedding, the `Load` class has no optimization variables and zero contribution to the objective. The load power varies with time and is defined by its `schedule` attribute. This schedule is read in from a file (e.g., Table VI) and placed in a `Schedule` class object. The power demand for a given time period can then be accessed by calling `schedule.get_energy(time)`.

Listing 2. Load class with shedding allowed

```
class Shedable_Load(OptimizationObject):
    def __init__(self,name,bus,schedule,
        cost_shedding=1000000):
        #load in inputs and set up
            optimization structure
        update_attributes(self,locals())
        self.init_optimization()
    def power(self,time): return self.
        get_variable('power',time)
    def shed(self,time): return self.
        schedule.get_energy(time) - self.
        power(time)
    def cost(self,time): return self.
        cost_shedding*self.shed(time)
    def create_variables(self,times):
        for time in times:
            self.add_variable('power','Pd
                ',time,low=0,high=self.
                schedule.get_energy(time))
    def create_objective(self,times):
        return sum(self.cost(time) for
        time in times)
```

Without shedding, the load power is simply a parameter. With shedding, as shown in the class `Shedable_Load` in Listing 2, the load power becomes a variable with a upper bound equal to the scheduled load amount. This variable is set up within `create_variables()` using `add_variable()` – both of which are standard methods for all `OptimizationObject` classes. Variables are stored within the class and after the problem is solved are set to the variable's solution value. From outside the `Shedable_Load` class, the power demand value is accessed through the `power` method, just as in the `Load` class without shedding.

With shedding, the load now makes a contribution to the total system cost (i.e., objective), if shedding occurs. This contribution is defined as difference between the variable's value and the scheduled amount (`shed`), multiplied by the (very high) penalty on load shedding. This contribution is defined within `create_objective()` (also a standard method for `OptimizationObject` classes).

Adding or modifying constraints, variables, and even system modes can be done easily and transparently by using the existing structure for optimization objects that exists in Minpower. No special mathematical optimization knowledge is required and the programming closely follows existing templates.

VII. Conclusion

The power systems community will benefit from an open source optimization toolkit. Minpower fills this role by using existing state-of-the-art open source software and a flexible, powerful, yet simple interface. Minpower can be used as a learning tool by students or as a research tool for modern power system problems. Tutorials, documentation, and the source code are available at minpowertoolkit.com.

Acknowledgements

The authors are grateful to ERCOT for providing hourly aggregate load and wind data that made the simulation in §VI-C possible. The authors would also like to thank the development team of Coopr for their support.

References

[1] "Task Force on Open Source Software for Power Systems." [Online]. Available: http://ewh.ieee.org/cmte/psace/CAMS_taskforce/

[2] R. Zimmerman, C. Murillo-Sánchez, and R. Thomas, "MATPOWER : Steady-State Operations, Planning, and Analysis Tools for Systems Research and Education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, 2011.

[3] R. Lincoln, "PyPower." [Online]. Available: www.pypower.com

[4] J. Sun and L. Tesfatsion, "An agent-based computational laboratory for wholesale power market design," in *IEEE Power Engineering Society General Meeting*, Jun. 2007.

[5] L. Vanfretti and F. Milano, "Application of the PSAT, an Open Source Software, for Educational and Research Purposes," in *IEEE Power Engineering Society General Meeting*, Jun. 2007.

[6] S. Cole and R. Belmans, "MatDyn, A New Matlab-Based Toolbox for Power System Dynamic Simulation," *IEEE Transactions on Power Systems*, vol. 26, no. 3, 2010.

[7] M. Zhou and S. Zhou, "Internet, Open-source and Power System Simulation," in *IEEE Power Engineering Society General Meeting*, Jun. 2007.

[8] F. Milano, "Dome." [Online]. Available: http://www3.uclm.es/profesorado/federico.milano/dome.htm

[9] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/

[10] "GLPK - GNU Linear Programming Kit." [Online]. Available: http://www.gnu.org/software/glpk/

[11] "SCIP: An open source MIP solver and constraint integer programming framework." [Online]. Available: http://scip.zib.de/

[12] R. Lougee-Heimer, "The common optimization interface for operations research: Promoting open-source software in the operations research community," *IBM Journal of Research and Development*, vol. 47, no. 1, January 2003.

[13] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: modeling and solving mathematical programs in Python," *Mathematical Programming Computation*, vol. 3, no. 3, Aug. 2011.

[14] M. Carrión and J. Arroyo, "A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem," *IEEE Transactions on Power Systems*, vol. 21, no. 3, Aug. 2006.

[15] Sandia National Labs, "Coopr Solvers." [Online]. Available: https://software.sandia.gov/trac/coopr/wiki/GettingStarted/Solvers

[16] W. E. Woodruff, J.-P. Watson, and D. L. Hart, "PySP : Modeling and Solving Stochastic Programs in Python," *Mathematical Programming Computation*, to be published.

[17] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, May-Jun 2007.

[18] M. Selvan and K. Swarup, "Development of power flow software using design patterns," *IEEE Transactions on Power Systems*, vol. 21, no. 2, 2006.

[19] "Sphinx - a Python Documentation Generator." [Online]. Available: http://sphinx.pocoo.org/

[20] A. J. Wood and B. F. Wollenberg, *Power Generation, Operation and Control*, 2nd ed. John Wiley & Sons, 1996.

[21] "Minpower Documentation - Creating a Problem." [Online]. Available: http://minpowertoolkit.com/creating-problems.html

[22] EPA, "eGrid." [Online]. Available: www.epa.gov/cleanenergy/energy-resources/egrid/

[23] ——, "Clean Air Markets Data." [Online]. Available: http://camddataandmaps.epa.gov/gdm/

[24] Northwest Power and Conservation Council, "Sixth Northwest Conservation and Electric Power Plan Appendices," Northwest Power and Conservation Council, Tech. Rep. February, 2010. [Online]. Available: http://www.nwcouncil.org/energy/powerplan/6/

[25] WECC, "WECC Transmission Expansion Planning Policy Committee 2020 Base Case Dataset." [Online]. Available: http://www.wecc.biz/committees/BOD/TEPPC/