Dynamic and Configurable Mathematical Modelling of a Hydropower plant

Research in Progress Paper

Michael Barry and René Schumann

Smart Infrastructure Laboratory
HES-SO Valais / Wallis ,
Rue de Technopôle 3, 3960 Sierre, Switzerland
michael.barry@hevs.ch | rene.schumann@hevs.ch

Abstract. Switzerland, among many other countries in Europe, are transitioning their energy economy away from nuclear power to renewable energies. One common, but key technology to make this transition possible is hydropower plants, of which there are 604 plants producing around 56% of Switzerland's energy production. To facilitate this transition we must modify the current operation of hydropower to compensate for recent developments in the energy market that have appeared due to increased use of other renewable energies. Mathematical modelling is a common method for planning the operation of a hydropower plant. However, mathematical models are know to be problem specific and static and therefore they are expensive to modify. In this paper we approach this problem through a modular configurable and dynamic design for the commonly used mathematical models and show how it can be implemented using the GAMS modelling language. We discuss an interactive configuration process and our research approach to automating it.

1 Introduction

Mathematical modelling of operating a hydropower (HP) plant is well studied and establish in industry [1–3]. Current models are highly accurate and have little room for improvements. However, the environment they operate in is changing drastically, forcing us to modify and adapt our models. The energy market is shifting from a push to a pull market, drastically challenging how a HP plant needs to operate. In a push market, the HP plant could produce as they see fit, adjusting to seasonal patterns. However, in a pull market, the HP plant needs to adjust according to the energy price. Due to renewable energies and their dependency on the weather, the energy price fluctuates. These fluctuations are more difficult to predict compared to seasonal fluctuations. In particular, such fluctuations cannot be predicted a year in advance, but rather required short term planning. In addition, the intra-day market has become a potential benefit for HP plants. The intra-day market allows producers to trade their energy in terms of 15 minute time slots. As the HP plants can transition from different

production levels quicker than other types of power plants, they have an inherent advantage in this market. There are also speculations that the time frame for the intra-day market may be reduced even further. These changes in the market environment force us to modify our models accordingly. In particular, we must be able to plan short term by having light models with a short runtime so that more current data can be used to more accurately predict the fluctuations.

This is a challenge, as the problems has become bigger due to the smaller time frame. In addition, flexibility in the production of a HP plant has become valuable due to high fluctuations in the energy price and therefore many HP plants are being modified to enable pump storage. However adjusting currently used operational models can become complex making analysis of the potential in such modifications difficult.

These issues call for mathematical models that are more flexible. In this paper we propose a dynamic and configurable design for the mathematical models that can be easily modified for current market needs. In particular it is possible to run simulations that vary greatly in their size and required runtime as well as their functionality from just one model through configuration. We propose a modular design in Section 2, analysis the mathematical needs as well as a GAMS implementation of these needs for such a model in Section 3, report the current state in development in Section 4 and then discuss our research direction that could build on our design in Section 5.

2 Modular Design

Mathematical models are known to be extremely problem specific and complex making it difficult to implement a dynamic and configurable model. We use a modularised approach, inspired by object orientated design, to overcome this difficulty. Each element in a model, such as a turbine or a market, is separated into modules. Each module then contains all variables and function for that given element. To remove an element from the model, one can simply exclude the module itself. For example, a module "markets" sums up the profits from each market. If a market is removed, this module still operates by summing up the profit from the remaining markets. As a result, it is still a valid model, but no logger utilises one of the markets. Adding or removing a module allows us to easily configure a model as shown in Fig 1.

We consider our design to be dynamic as it can be easily extended. The initial model is extremely simple and only uses simplified representations of the real world. We use a dynamic design to incrementally expand the model to incorporate further constraints, building a more accurate representation of the real world in the process.

We consider our design to be configurable as it can be easily configured for individual case studies. A model is a general design which we then configure to simulate individual HP Plants. For example, different HP plants use different turbines and therefore we must be able to easily switch the turbine used in our

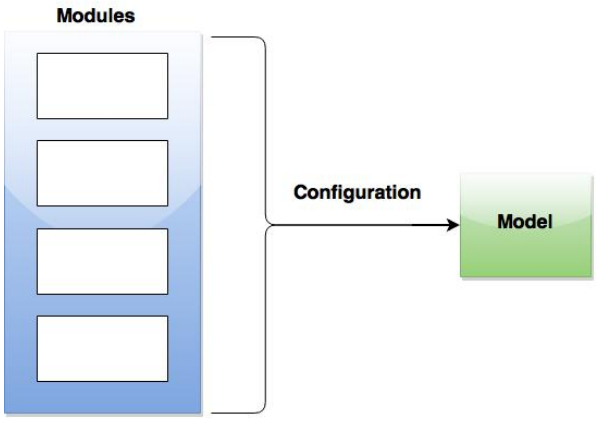

Fig. 1. Graphical representation of how a set of modules form a configuration.

model through a simple interface rather than by redesigning our model. This allows us to simulate many HP plants with just one model.

Our design also gives a level of abstraction that can help keep an overview of the model and is particular useful if a a model is designed by several developers, as it allows developers to work on individual modules without knowing about or interfering with the work on another module. In figure 2 we show the design of the model.

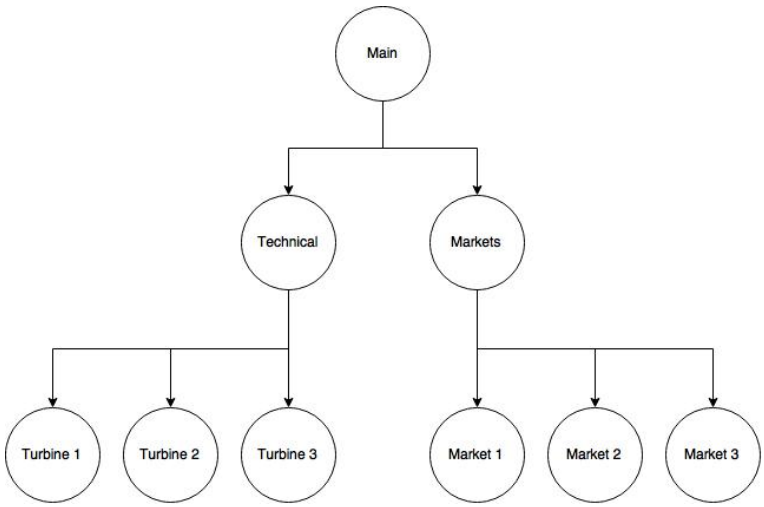

Fig. 2. Graphical representation of the models design with each modules dependencies.

2.1 Dependencies

To configure a model, one must simply define which modules are to be used. The set of modules that are used is said to be a configuration. However, not all configurations are said to be valid. For example, *market 1* is dependant on the module and the *markets* module is dependant on the *main* module. Therefore, a configuration using *market 1* must have the modules *markets* and *main* also enabled. We can also define these dependencies in a logical mannerism, such as *market 1 implies market 2*, or more formally as shown below:

$$market1 \rightarrow markets$$

The complete set of dependencies can therefore be defined as shown below

$$\begin{gathered} markets \rightarrow main \\ market1 \rightarrow markets \\ market2 \rightarrow markets \\ market3 \rightarrow markets \\ technical \rightarrow main \\ turbine1 \rightarrow technical \\ turbine2 \rightarrow technical \\ turbine3 \rightarrow technical \end{gathered}$$

2.2 Conditionals

In addition to the dependencies, modules may have to consider additional conditionals that arise from the physical world. For example we wish to only use one turbine and therefore only one *turbine* module may be active in a configuration. In such a case we describe the relationship to it's parent module with an $\oplus$ (exclusive OR) conditional, as a valid configuration must have only one of the child modules active. It is is similar to the $\oplus$ used in logic gate, except that we consider that there can be several inputs, which is the equivalent of several $\oplus$ combined. The truth table is given below for three inputs. Note that if only one child node exists, it implies that the child module must always be active in a valid configuration and therefore the parent and child node are dependant on each other.

A	B	C	XOR
F	F	F	F
F	F	T	T
F	T	F	T
T	F	F	T
F	T	T	F
T	F	T	F
T	T	F	F
T	T	T	F

In other cases where any combination of the modules exist we can describe the relationship as a tautology, as any combination of the related modules can be considered as true. Therefore on logical terms it can be ignored.

If we consider the $\oplus$ conditional in our design diagram, we can graphically describe the validity logic as shown in Figure 3.

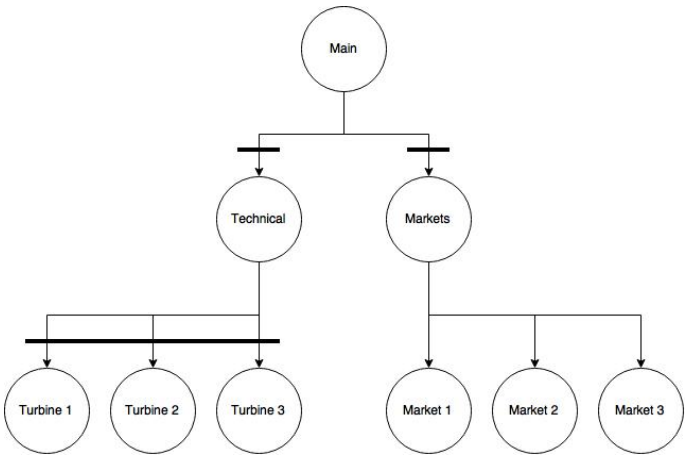

Fig. 3. Graphical representation of the logic. Lines represent dependencies and lines with a bar across symbolise the XOR operator.

To express the validity logic formally, we first describe the $\oplus$ conditionals as shown below.

$$main \rightarrow technical$$
$$technical \rightarrow turbine1 \oplus turbine2 \oplus turbine3$$
$$main \rightarrow markets$$

We must then add our $\oplus$ conditionals to the dependencies defined previously, which can then be simplified to:

$$markets \leftrightarrow main$$
$$market1 \rightarrow markets$$
$$market2 \rightarrow markets$$
$$market3 \rightarrow markets$$
$$technical \leftrightarrow main$$
$$technical \leftrightarrow turbine1 \oplus turbine2 \oplus turbine3$$

This design provides a level of abstraction from the underlying mathematical model, allowing us to configure the model without requiring understanding of the mathematical model. However, to implement the design, we must consider how the logical relationships can be realised in mathematical terms. The mathematical model is described in the next Section.

3 Mathematical Model

The mathematical model describes the environment in which the HP plant operates in mathematical terms. Typically, a model consists of variables, constraints (functions) and an objective function. For this paper we focus on the constraints and what is required in mathematical terms to implement our modular design.

We identify three operations that are required for a dynamic and configurable design:

- Add function
- Replace function
- Aggregate function.

These operations are used to modify the model in an iterative way without the need of redesigning previously built components. Below, in equations 1 - 6, a simple mathematical representation of our Model is given and will be used as an example.

$$\max. \sum_{i,m} P_{i,m} Q_{i,m} \tag{1}$$

$$Q_{i,m} = R_{i,m} \alpha \tag{2}$$

$$S_i = S_{i-1} + I_i - \sum_m R_{i,m} \tag{3}$$

$$S_i \leq S_{\max} \tag{4}$$

$$S_i \geq S_{\min} \tag{5}$$

$$\sum_m Q_{i,m} \leq Q_{max} \tag{6}$$

Where $P_{i,m}$ in equation 1 is the price at time interval i for market m, $Q_{i,m}$ in equation 1 is the produced energy for time interval i and market m, $R_{i,m}$ in equation 2 is the water released from the reservoir at time interval i for market m, α in equation 2 is the efficiency of the turbine (the amount of energy produced per water used), S_i in equation 3 is the storage level at time interval i, I_i in equation 3is the inflow of water into the reservoir at time interval i, $S_{\max}$ in equation 4 is the maximum storage level of the reservoir, $S_{\min}$ in equation 5 is the minimum storage level of the reservoir and Q_{max} in equation 6 is the maximum production level.

3.1 Add function

The *add function* operation is used in the mathematical module to enable the operations required to add a module and represents a tautology. Using this method we can add or remove a module (or constraint) without breaking the model. Mathematically, all it requires is the definition of another function. For example, the function below is added, constricting how much water must be left in the reservoir for the last time interval t_{final}.

$$S_{t_{max}} \geq S_{final} \tag{7}$$

Where t_{max} is the last time interval of t and S_{final} is the final storage level at t_{max}.

We use the General Algorithmic Modelling Software (GAMS) as a modelling language. Each module is defined in a separate file. We define the above constraint in a file *initial_storage_constraint.gms* as such:

```
Equation
final_storage (t)
;

final_storage(t)..
storage_final        =l=        storage(t) $(ORD(t) eq CARD(t))
;
```

The module can then be included in the model by adding the following include statement to the *main.gms* file.

```
$INCLUDE initial_storage_constraint
```

3.2 Replace function

The *replace function operation* allows for a a constraint to be replaced and is used to implement the $\oplus$ conditional. When multiple implementations of a constraint exist, we use this operation to force the model to only accept one implementation. Mathematically, this is done by simply replacing a function. For example, the reservoir may lose water through other effects such as evaporation. This affect can be added by replacing equation 3 with equation 8 to accurately represent the water loss:

$$S_i = S_{i-1} + I_i - \sum_m R_{i,m} - L_i \tag{8}$$

Where L_t is the water loss at time interval i.

In GAMS we can implement this by implementing the *storage* module called *storage.gms* as below:

```
Equation
storage (i)
;

storage(i)..
storage(i)          =e=       storage(i-1) + inflow(i)
                            - sum(m, release(i,m))
```

a file *storage_evap.gms* is then implemented using the modified constraint:

```
Equation
storage (i)
;

storage(i)..
storage(i)          =e=       storage(i-1) + inflow(i)
                            - sum(m, release(i,m))
                            - loss(i)
```

We can then include the modules by removing, or commenting out, the include statement for the *storage* module and replace it with a new include statement:

```
*$INCLUDE storage
$INCLUDE storage_evap
```

3.3 Aggregate function

Modules that relate to each other as a tautology can also be implemented using the *aggregate function* operator. Rather than adding a constraint as the *add function* operator does, it uses an aggregate function that then allows each element in that function to be modified by a separate module. For example, each market on which a HP plant can trade on is defined in separate functions contained within a separate module. It can trade on any number of the defined markets and it is important that the model can exclude a market without having to redefine the function that calculates the total income from all markets, which is given in equation 1. Equation 1 sums up the income from each market. We introduce a new variable X to represent the income and can therefore redefine equation 1 as such:

$$\max. \sum_{i,m} X_{i,m} \tag{9}$$

We can then define the functionality for each market separately:

$$X_{i,1} = P_{i,1} Q_{i,1} \tag{10}$$

$$X_{i,2} = P_{i,2} C_{i,2} \tag{11}$$

Where $C_{i,1}$ is a capacity bid. We can exclude markets by replacing the income functions with a zero value as such:

$$X_{i,1} = 0 \tag{12}$$

The zero value is then ignored within the sum function. In GAMS, we can implement the markets module in file *markets.gms* as such:

```
Equation
income_total (i)
;

income_total(i)..
income_total(i)          =e= sum(m, income(i,m))
```

Market1 can then be implemented in a file called *market1.gms*:

```
Equation
income(i)
;

income(i)..
income(i)          =e= price(i,1) * production(i,1)
```

Market2 is implemented in a file called *market2.gms*:

```
Equation
income(i)
;

income(i)..
income(i)          =e= price(i,2) * production(i,2)
```

Once these files are implemented, a market can be added or removed using the following include statement in the *main.gms* file:

```
$INCLUDE markets.gms

$INCLUDE market1.gms
*$INCLUDE market2.gms
```

As the include statement for *market 2* is commented out, only *market 1* would be considered. As GAMS automatically assigns a zero value to the income variable for *market 2*, it would be ignored within the sum function in *markets.gms*.

Using these three operations we are able to iteratively expand the model during development. In addition, we can remove these expansions again without the rest of the model failing. This allows flexibility and opens up to the possibility of a configurable and dynamic design.

4 Configuration

A defined set of active modules is considered to be a configuration and the above logic defines the validity of such a configuration. We can then consider all valid configurations as the search space for a configuration problem. As now we have a defined search space, an ideal configuration must be selected. In the initial stage we simply use an interactive configuration process in which a user can select the desired configuration through an interface.

In this interface we can define a configuration, which defines further conditions, such as *market1 is true*. Once these are defined and combined with the previous logic, we can check the validity with our logic to verify the validity of our configuration. This in turn is fed back to the interface to help guide the user to valid configurations.

However, we wish to move towards an automatic configuration process based on two objectives: the runtime and the scale of the model. The required time to solve the model (runtime) greatly depends on the configuration. Using our method, it is possible to configure an extremely basic and easy to solve model or a large but more realistic model. To facilitate this we must know more of the relationship between the model's size and it's runtime. We have implemented a multi objective evolutionary algorithm based on the SPEA2 algorithm [6] to explore this relationship by exploring the search space of all valid configurations using runtime and the size of the model as objectives. This generates a Pareto front of the objectives, showing a clear relationship and is described in our paper submitted to the Energy Informatics conference [5]. However, as the model still needs to be extended before relevant understanding of the search space can be gained, we discuss this and other further work in Section 6.

5 Current State

Currently we have implemented a simple model using our modular design and we have implemented a Java application to configure the model. Through adding output modules to the model, the Java application is also able to automatically find all relevant output files and produce various 2D and 3D charts. This can be used to display variable values graphically and can help in understanding the behaviour of a model or identifying any problems. The evolutionary algorithm has already shown that our design works and can show a simple relationship between the size of the model and the runtime. The model itself is still currently very simple and consists only of around 20 modules. However, it is expected that the model will be expanded to a much larger size as part of our project in the context of the SNSF supported project NRP70 *Energy Turnaround*. To test our model we have organised two case studies with our project partners in Switzerland, who have kindly provided the required data.

6 Research roadmap

The aim of our research is to develop methods for configuring mathematical models that are based on a modular design. In particular, we develop methods that considers the relationship between the runtime and the complexity of the model. As one would expect, there is a direct relationship between the complexity of a model and the runtime. Our evolutionary algorithm based on our basic model is already capable of showing that. However, we wish to expand the configuration search space by extending our mathematical model and test weather this basic relationship still exists, or whether more advanced patterns such as the easy-hard-easy pattern [4] can be observed. Such knowledge is valuable in the configuration process, as configurations are then identified which have a relatively high complexity and a low runtime, giving us a fast and accurate mathematical models.

In addition, we will analyse the stability of the search space, as mathematical models are prone to be unstable and therefore small changes in the model can often have a large impact on the runtime. The stability represents whether similar configurations also have a similar runtime. The stability of the search space is greatly dependant on the variables we use to measure the runtime and the complexity. The runtime is fairly simple to measure accurately. However, the complexity is more difficult. Several measurements exist and a feature selection process may be required to select what measurements best represent the problems complexity. A stable search space is essential for the configuration process as it would allow us to use methods beyond a simple interface, allowing us to move from an interact configuration process to an automated configuration process. Iterative methods, such as the evolutionary algorithm developed, require a a stable search space.

In addition, we wish to use methods that can use the knowledge of the search space to quickly and accurately predict the runtime of a model without having to run it. This would require methods for capturing the knowledge gained from the search space exploration and using it to predict the runtime of a configuration. Once we show that the relationship between the complexity and runtime can be established, we can make a more formal analysis of what effects the runtime and whether it is linked to which of the mathematical operations we used to connect the modules with.

From a developers perspective we also wish to extend our current design and develop a method for automatically creating the validity logic, whether by trial and error methods or utilising comment within the code of the mathematical model. As we would simply need to test if the model compiles, this could be generated quickly and would save the developer the need to generate the logic manually

Acknowledgements

This project is supported by SNSF as part of the National Research Programme NRP70 *Energy Turnaround*.

We also wish to acknowledge the work from our project partners Prof. Dr. Hannes Weight and his PhD student Moritz Schillinger from Basel University who support us on modelling the markets.

References

1. Alfieri, L., Perona, P., Burlando, P.: Optimal water allocation for an alpine hydropower system under changing scenarios. Water Resources Management 20(5), 761–778 (2006), http://dx.doi.org/10.1007/s11269-005-9006-y
2. Guo, S., Chen, J., Li, Y., Liu, P., Li, T.: Joint operation of the multi-reservoir system of the three gorges and the qingjiang cascade reservoirs. Energies 4(7), 1036–1050 (2011), http://www.mdpi.com/1996-1073/4/7/1036
3. Álvaro Jaramillo Duque, Castronuovo, E.D., Sánchez, I., Usaola, J.: Optimal operation of a pumped-storage hydro plant that compensates the imbalances of a wind power producer. Electric Power Systems Research 81(9), 1767 – 1777 (2011), http://www.sciencedirect.com/science/article/pii/S0378779611000940
4. Mammen, D.L., Hogg, T.: A new look at the easy-hard-easy pattern of combinatorial search difficulty. JAIR 7, 47–66 (1997)
5. Michael Barry, Moritz Schillinger, H.W., Schumann, R.: Configuration of hydro power plant mathematical models. In: Submitted to Energieinformatik 2015 (2015)
6. Zitzler, E., Laumanns, M., Thiele, L.: Spea2: Improving the strength pareto evolutionary algorithm. Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK) (2001)