# I - Overview of the differences between SpineOpt and Backbone

In general, Backbone is more specific than SpineOpt: there is a tendency to use more specific parameters for particular cases, whereas SpineOpt often uses more generic parameters and functions.

For example:

* In Backbone there are specific objects for managing emissions, whereas in SpineOpt nodes are used in a more general way. Thus, one defines an emission\_content in Backbone, but uses a fix\_ratio\_out\_in\_flow on a classic node in SpineOpt
* In Backbone, certain objects and parameters are specific to fuels, whereas in SpineOpt, the same elements are managed as standard nodes.
* In Backbone, the distinctions between "price", "priceChange", "emissionPrice" and "emissionPriceChange" are specifically marked but are managed in a more general way by a single "node\_slack\_penalty" in SpineOpt: there is no specific price parameter.
* In Backbone there are inputs and outputs, which don’t exist (or tend to disappear) in SpineOpt

Moreover, some specificities must not be forgotten when creating databases in Backbone.

* For example, ffLevels, effSelectors and node groups need to be carefully specified. Errors in these definitions can cause problems if incorrectly specified, missing or associated with the wrong types of reserves, the wrong nodes, etc.
* In Backbone, the definition of many parameters requires activation flags: those are defined by a value and activated by another binary parameter. If this second parameter is not activated, the associated value is not considered. In addition, there are 'flags' to determine the use of a time series for data, which is not the case in SpineOpt. These elements must be considered when creating a database in Backbone.

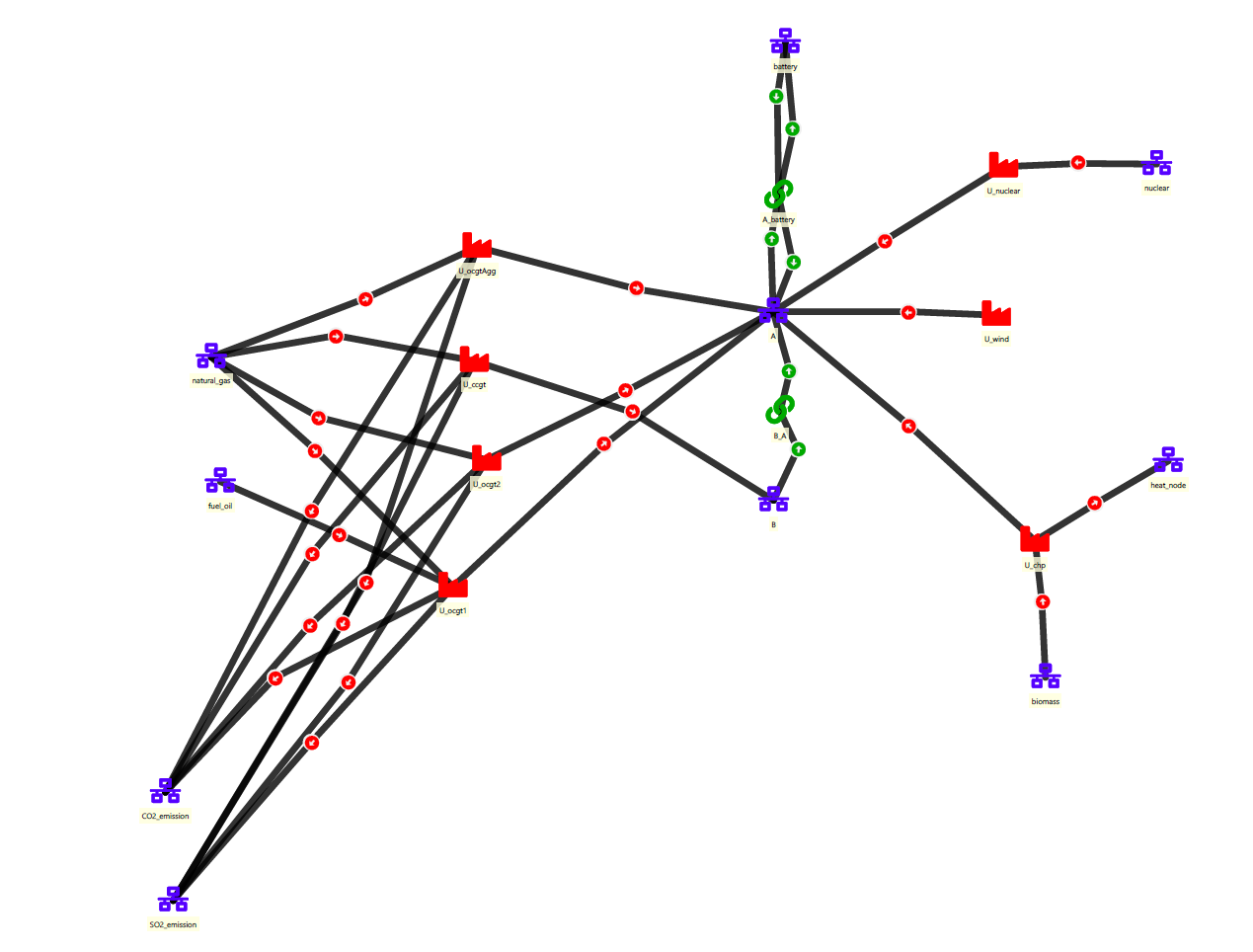
Finally, a lot more things are defined through relationships in Backbone than in SpineOpt (for example, most of the parameters than one defines on nodes in SpineOpt are defined on grid\_nodes in Backbone). Because of this kind of differences, some things are not directly translatable from one model to another, because it only depends on one object in SpineOpt, but depends on several in Backbone. For example, efficiencies only depend on units in Backbone, but depend on the unit, the input, and the output in SpineOpt.

There can be several ways of interpreting a notion of Backbone in SpineOpt, because generic parameters can be used in different ways. On the other hand, translating from SpineOpt to Backbone is complicated because of the high number and specificity of parameters, objects, relationships, etc.

The automation of translation between the two model formats seems therefore complicated. The use of a generic databases format, translated in both model formats, would probably be much easier and more practical than direct translation.

# II – Used system

The aim was to translate from Backbone to SpineOpt a system that was both as simple as possible and at the same time included many different features to determine the points of difficulty in a potential translation between the 2 data formats of the 2 models.



The system is made up of 6 units, plus an aggregated unit: 1 nuclear unit, 1 wind unit, 2 Open Cycle Gas Turbine (OCGT) unit, 1 Combined Cycle Gas Turbine (CCGT) unit and a Combined Heat and Power (CHP) unit. The aggregated unit combines the 2 OCGT units.

The system includes 2 electricity nodes, a battery, CO2 and SO2 emissions, a heat node, as well as nodes representing resources: nuclear, fuel oil, natural gas and biomass.

All the units, apart from the CCGT unit, are connected to electricity node A, to which they supply their electricity production. The battery is also connected to node A. The CCGT unit is connected to node B. Nodes A and B are connected by a link running from B to A.

All units that use oil or gas produce CO2 and SO2 emissions, while CHP, wind and nuclear units produce no emissions.

# III – Observations

General observations :

* GAMS does not record 0s: it is not possible to distinguish between a 0 value and no value. This has certain consequences that need to be taken into account:
  + the results have "holes" instead of 0s, which need to be filled in ;
  + it is sometimes necessary to distinguish between a "true" 0 and a value that does not translate, when switching from Backbone to SpineOpt.
* Demands in SpineOpt are counted positively towards the node (since they are demands). Influxes in Backbone are counted negatively towards the node (values towards the outside the node are counted positively)
* There are many parameters that do not need to be defined in SpineOpt when the default value is the correct one, but which must be defined in Backbone for everything to work correctly.
* If things are not defined correctly in Backbone, it often happens that there is no error, but the behaviour is not what is expected. An example: if one doesn't specify the efficiency of a unit, it is considered ideal, but the unit no longer consumes fuel. It then produces energy from nothing, and simply satisfies the demand.
* Be careful with the format of Map type data in Backbone: one generally needs to specify the time index and the forecast, but some parameters only require the time index: adding a forecast is then equivalent to deactivating this parameter. Example: one doesn't need to specify a specific forecast for priceChange. However, one does need to specify a forecast for influx.
* The results are not saved in the same format. One therefore needs to reformat them to compare them. I have coded a first version of a code to do this automatically.

Additional observations by object classes :

* connections: in SpineOpt, connection is an object class in its own right, in Backbone it's just a relationship. So things have to be defined differently. In particular, in SpineOpt, one needs to define an input capacity and an output capacity, whereas in Backbone, it's a single relationship with a capacity, and possibly a dischargeLoss etc. In addition, in SpineOpt, one needs to define an input capacity and an output capacity. Furthermore, in SpineOpt one defines the input/output ratio, whereas in Backbone a loss ratio is defined.
* units :
  + there is no equivalent to the UnitSize parameter in SpineOpt, only unit\_capacity. In Backbone, there are both. This sometimes creates differences in modelling, see next point.
  + unit efficiencies are not defined in the same way in Backbone and SpineOpt. In Backbone, a table of parameters and a relationship between the unit and a certain effLevel allow to define complex efficiencies. In SpineOpt, one can translate the simplest efficiencies (for example directOnLP effLevel, linear) by defining incremental\_heat\_rate and idle\_heat\_rate. To do this, one needs to obtain the slope and section of the efficiencies defined in Backbone. The incremental\_heat\_rate parameter then corresponds to the slope multiplied by the UnitSize defined in Backbone (and not defined in SpineOpt, see previous point); idle\_heat\_rate corresponds to the cross-section multiplied by the UnitSize. The minimum\_operating\_point can be translated as is in SpineOpt.
* user\_constraint: this is an example where it is possible to translate something in several ways from Backbone to SpineOpt. The constraint on the ratio of CHP unit outputs is here either translatable through a user\_constraint similar to the Backbone model, or through the definition of ratios on unit\_node\_node relationships.

# IV – Additional comments

Our translation process with the 6-unit system was of course not exhaustive. In particular, it should be noted that we have not made use of any advanced features linked to reserves. Reserves management in Backbone is fairly complex, and probably difficult to translate to SpineOpt. Again, translating from a generic data format to SpineOpt and Backbone is probably easier than translating directly from one to another.