**Important:** This document resumes all the differences between the [original TEMOA source](https://github.com/TemoaProject/temoa) code and the TEMOA version by [MAHTEP Group](https://github.com/MAHTEP/TEMOA).

**ENVIRONMENT**

The “environment.yml” file is modified as following:

* Delete the “- coincbc” row if Windows is used.
* Add “- gurobi” among channels and dependencies.

**BARRIER ALGORITHM**

The following row was added at row 381 of **temoa\_run.py** (this is to select the barrier algorithm for the solution of the optimization problem, resulting to be more efficient for complex models):

|  |
| --- |
| self.optimizer.set\_options('Method=2') # Barrier Algorithm |

**EMISSION ACTIVITY**

Create a new table in the **database** with the following structure:

|  |
| --- |
| CREATE TABLE "CommodityEmissionFactor" (  "input\_comm" text,  "emis\_comm" text,  "ef" real,  "emis\_unit" text,  "ef\_notes" text,  PRIMARY KEY("input\_comm","ef","emis\_comm"),  FOREIGN KEY("input\_comm") REFERENCES "commodities"("comm\_name"),  FOREIGN KEY("emis\_comm") REFERENCES "commodities"("comm\_name")  ); |

Substitute in **temoa\_model.py** row 221 with the following:

|  |
| --- |
| M.EmissionActivity = Param(M.EmissionActivity\_reitvo, default=0) |

**SERVICE DEMANDS PROJECTION**

Create new tables in the **database** with the following structure:

|  |
| --- |
| CREATE TABLE "Driver" (  "regions" text,  "periods" integer,  "driver\_name" text,  "driver" real,  "driver\_notes" text,  PRIMARY KEY("regions", "periods", "driver\_name"),  FOREIGN KEY("regions") REFERENCES "regions"("regions"),  FOREIGN KEY("periods") REFERENCES "time\_periods"("t\_periods")  );  CREATE TABLE "Allocation" (  "regions" text,  "demand\_comm" text,  "driver\_name" text,  "allocation\_notes" text,  PRIMARY KEY("regions", "demand\_comm", "driver\_name"),  FOREIGN KEY("regions") REFERENCES "regions"("regions"),  FOREIGN KEY("demand\_comm") REFERENCES "commodities"("comm\_name"),  FOREIGN KEY("driver\_name") REFERENCES "Driver"("driver\_name"),  );  CREATE TABLE "Elasticity" (  "regions" text,  "periods" integer,  "demand\_comm" text,  "elasticity" real,  "elaticity\_notes" text,  PRIMARY KEY("regions", "periods", "demand\_comm"),  FOREIGN KEY("regions") REFERENCES "regions"("regions"),  FOREIGN KEY("periods") REFERENCES "time\_periods"("t\_periods"),  FOREIGN KEY("demand\_comm") REFERENCES "commodities"("comm\_name")  ); |

**CAPACITY FACTOR**

Create a new table in the **database** with the following structure:

|  |
| --- |
| CREATE TABLE "CapacityFactor" (  "regions" text,  "tech" text,  "vintage" integer,  "cf" real,  "cf\_notes" text,  PRIMARY KEY("regions","tech","vintage"),  FOREIGN KEY("tech") REFERENCES "technologies"("tech"),  FOREIGN KEY("vintage") REFERENCES "time\_periods"("t\_periods")  ); |

Add to **temoa\_config.py** at row 157 the following rows:

|  |
| --- |
| ['param','CapacityFactor', '', '', 3], |

Add to **temoa\_initialize.py** at row 798 the following rows:

|  |
| --- |
| def CapacityFactorIndices(M):  indices = set(  (r, t, v)  for r in M.regions  for t in M.tech\_all  for v in M.vintage\_all  )  return indices |

Add to **temoa\_model.py** at row 141 the following rows:

|  |
| --- |
| M.CapacityFactor\_rtv = Set(dimen=3, initialize= CapacityFactorIndices)  M.CapacityFactor = Param(M.CapacityFactor\_rtv, default=1) |

Add to **temoa\_rules.py** at row 82 the following row:

|  |
| --- |
| \* value(M.CapacityFactor[r, t, v]) \ |

Add to **temoa\_rules.py** at row 91 the following row:

|  |
| --- |
| \* value(M.CapacityFactor[r, t, v]) \ |

Add to **temoa\_rules.py** at row 133 the following row:

|  |
| --- |
| \* value(M.CapacityFactor[r, t, v]) \ |

**GROUP CONSTRAINTS**

Delete the tables “MinGenGroupWeight” and “MinGenGroupTarget” from the **database** and create new tables with the following structure:

|  |
| --- |
| CREATE TABLE "TechGroupWeight" (  "regions" text,  "tech" text,  "group\_name" text,  "weight" real,  "tech\_desc" text,  PRIMARY KEY("tech","group\_name","regions")  );  CREATE TABLE "MinActivityGroup" (  "periods" integer,  "group\_name" text,  "min\_act\_g" real,  "notes" text,  PRIMARY KEY("periods","group\_name")  );  CREATE TABLE "MaxActivityGroup" (  "periods" integer,  "group\_name" text,  "max\_act\_g" real,  "notes" text,  PRIMARY KEY("periods","group\_name")  );  CREATE TABLE "MinCapacityGroup" (  "periods" integer,  "group\_name" text,  "min\_cap\_g" real,  "notes" text,  PRIMARY KEY("periods","group\_name")  );  CREATE TABLE "MaxCapacityGroup" (  "periods" integer,  "group\_name" text,  "max\_cap\_g" real,  "notes" text,  PRIMARY KEY("periods","group\_name")  );  CREATE TABLE "MinInputGroup" (  "regions" text,  "periods" integer,  "input\_comm" text,  "group\_name" text,  "gi\_min" real,  "gi\_min\_notes" text,  FOREIGN KEY("group\_name") REFERENCES "groups"("group\_name"),  FOREIGN KEY("input\_comm") REFERENCES "commodities"("comm\_name"),  FOREIGN KEY("periods") REFERENCES "time\_periods"("t\_periods"),  PRIMARY KEY("regions","periods","input\_comm","group\_name")  );  CREATE TABLE "MaxInputGroup" (  "regions" text,  "periods" integer,  "input\_comm" text,  "group\_name" text,  "gi\_max" real,  "gi\_max\_notes" text,  FOREIGN KEY("group\_name") REFERENCES "groups"("group\_name"),  FOREIGN KEY("input\_comm") REFERENCES "commodities"("comm\_name"),  FOREIGN KEY("periods") REFERENCES "time\_periods"("t\_periods"),  PRIMARY KEY("regions","periods","input\_comm","group\_name")  );  CREATE TABLE "MinOutputGroup" (  "regions" text,  "periods" integer,  "output\_comm" text,  "group\_name" text,  "go\_min" real,  "go\_min\_notes" text,  FOREIGN KEY("group\_name") REFERENCES "groups"("group\_name"),  FOREIGN KEY("output\_comm") REFERENCES "commodities"("comm\_name"),  FOREIGN KEY("periods") REFERENCES "time\_periods"("t\_periods"),  PRIMARY KEY("regions","periods","output\_comm","group\_name")  );  CREATE TABLE "MaxOutputGroup" (  "regions" text,  "periods" integer,  "output\_comm" text,  "group\_name" text,  "go\_max" real,  "go\_max\_notes" text,  FOREIGN KEY("group\_name") REFERENCES "groups"("group\_name"),  FOREIGN KEY("output\_comm") REFERENCES "commodities"("comm\_name"),  FOREIGN KEY("periods") REFERENCES "time\_periods"("t\_periods"),  PRIMARY KEY("regions","periods","output\_comm","group\_name")  ); |

Delete rows 216-217 (“MinCapacitySum” and “MaxCapacitySum”), 471-477 (“MaxCapacitySet”) and 478-485 (“MinCapacitySet”) from **temoa\_model.py**.

Delete rows 1875-1889 (“MaxCapacitySet\_Constraint”) and 1894-1908 (“MinCapacitySet\_Constraint”) from **temoa\_rules.py**.

Substitute in **temoa\_config.py** rows 131-132 with:

|  |
| --- |
| ['param','TechGroupWeight', '', '', 3],  ['param','MinActivityGroup', '', '', 2],  ['param','MaxActivityGroup', '', '', 2],  ['param','MinCapacityGroup', '', '', 2],  ['param','MaxCapacityGroup', '', '', 2],  ['param','MinInputGroup', '', '', 4],  ['param','MaxInputGroup', '', '', 4],  ['param','MinOutputGroup', '', '', 4],  ['param','MaxOutputGroup', '', '', 4], |

Substitute in **temoa\_model.py** rows 223-224 with:

|  |
| --- |
| M.TechGroupWeight = Param(M.RegionalIndices, M.tech\_groups, M.groups, default=1)  M.MinActivityGroup = Param(M.time\_optimize, M.groups)  M.MaxActivityGroup = Param(M.time\_optimize, M.groups)  M.MinCapacityGroup = Param(M.time\_optimize, M.groups)  M.MaxCapacityGroup = Param(M.time\_optimize, M.groups)  M.MinInputGroup = Param(M.regions, M.time\_optimize, M.commodity\_physical, M.groups)  M.MaxInputGroup = Param(M.regions, M.time\_optimize, M.commodity\_physical, M.groups)  M.MinOutputGroup = Param(M.regions, M.time\_optimize, M.commodity\_physical, M.groups)  M.MaxOutputGroup = Param(M.regions, M.time\_optimize, M.commodity\_physical, M.groups) |

Substitute in **temoa\_model.py** rows 450-456 with:

|  |
| --- |
| M.MinActivityGroup\_pg = Set(  dimen=2, initialize=lambda M: M.MinActivityGroup.sparse\_iterkeys()  )  M.MinActivityGroupConstraint = Constraint(  M.MinActivityGroup\_pg, rule=MinActivityGroup\_Constraint  )    M.MaxActivityGroup\_pg = Set(  dimen=2, initialize=lambda M: M.MaxActivityGroup.sparse\_iterkeys()  )  M.MaxActivityGroupConstraint = Constraint(  M.MaxActivityGroup\_pg, rule=MaxActivityGroup\_Constraint  )  M.MinCapacityGroupConstraint\_pg = Set(  dimen=2, initialize=lambda M: M.MinCapacityGroup.sparse\_iterkeys()  )  M.MinCapacityGroupConstraint = Constraint(  M.MinCapacityGroupConstraint\_pg, rule=MinCapacityGroup\_Constraint  )  M.MaxCapacityGroupConstraint\_pg = Set(  dimen=2, initialize=lambda M: M.MaxCapacityGroup.sparse\_iterkeys()  )  M.MaxCapacityGroupConstraint = Constraint(  M.MaxCapacityGroupConstraint\_pg, rule=MaxCapacityGroup\_Constraint  )  M.MinInputGroup\_Constraint\_rpig = Set(  dimen=4, initialize=lambda M: M.MinInputGroup.sparse\_iterkeys()  )  M.MinInputGroupConstraint = Constraint(  M.MinInputGroup\_Constraint\_rpig, rule=MinInputGroup\_Constraint  )  M.MaxInputGroup\_Constraint\_rpig = Set(  dimen=4, initialize=lambda M: M.MaxInputGroup.sparse\_iterkeys()  )  M.MaxInputGroupConstraint = Constraint(  M.MaxInputGroup\_Constraint\_rpig, rule=MaxInputGroup\_Constraint  )  M.MinOutputGroup\_Constraint\_rpig = Set(  dimen=4, initialize=lambda M: M.MinOutputGroup.sparse\_iterkeys()  )  M.MinOutputGroupConstraint = Constraint(  M.MinOutputGroup\_Constraint\_rpig, rule=MinOutputGroup\_Constraint  )  M.MaxOutputGroup\_Constraint\_rpig = Set(  dimen=4, initialize=lambda M: M.MaxOutputGroup.sparse\_iterkeys()  )  M.MaxOutputGroupConstraint = Constraint(  M.MaxOutputGroup\_Constraint\_rpig, rule=MaxOutputGroup\_Constraint  ) |

Substitute in **temoa\_rules.py** rows 1768-1812 with:

|  |
| --- |
| def MinActivityGroup\_Constraint(M, p, g):  r"""  The MinActivityGroup constraint sets a minimum activity limit for a user-defined  technology group. Each technology within each group is multiplied by a  weighting function (:math:`MGW\_{r,t}`), which determines the technology activity  share that can count towards the constraint.  .. math::  :label: MinActivityGroup  \sum\_{S,D,I,T,V,O} \textbf{FO}\_{p, s, d, i, t, v, o} \cdot MGW\_{t|t \not \in T^{a}}  + \sum\_{I,T,V,O} \textbf{FOA}\_{p, i, t \in T^{a}, v, o} \cdot MGW\_{t \in T^{a}}  \ge MGT\_{p, g}  \forall \{p, g\} \in \Theta\_{\text{MinActivityGroup}}  where :math:`g` represents the assigned technology group and :math:`MGT\_r`  refers to the :code:`MinActivityGroup` parameter.  """  activity\_p = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g]  for r in M.RegionalIndices  for S\_t in M.tech\_groups if (S\_t not in M.tech\_annual) and ((r, p, S\_t) in M.processVintages.keys())  for S\_v in M.processVintages[r, p, S\_t]  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  activity\_p\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g]  for r in M.RegionalIndices  for S\_t in M.tech\_groups if (S\_t in M.tech\_annual) and ((r, p, S\_t) in M.processVintages.keys())  for S\_v in M.processVintages[r, p, S\_t]  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  min\_act = value(M.MinActivityGroup[p, g])  expr = activity\_p + activity\_p\_annual >= min\_act  return expr  def MaxActivityGroup\_Constraint(M, p, g):  r"""  The MaxActivityGroup constraint sets a maximum activity limit for a user-defined  technology group. Each technology within each group is multiplied by a  weighting function, which determines what technology activity share can count  towards the constraint.  .. math::  :label: MaxActivityGroup  \sum\_{S,D,I,T,V,O} \textbf{FO}\_{p, s, d, i, t, v, o} \cdot WEIGHT\_{t|t \not \in T^{a}}  + \sum\_{I,T,V,O} \textbf{FOA}\_{p, i, t, v, o} \cdot WEIGHT\_{t \in T^{a}}  \le MGGL\_{p, g}  \forall \{p, g\} \in \Theta\_{\text{MaxActivityGroup}}  where :math:`g` represents the assigned technology group and :math:`MGGL`  refers to the :code:` MaxActivityGroup` parameter.  """  activity\_p = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g]  for r in M.RegionalIndices  for S\_t in M.tech\_groups if (S\_t not in M.tech\_annual) and ((r, p, S\_t) in M.processVintages.keys())  for S\_v in M.processVintages[r, p, S\_t]  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  activity\_p\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g]  for r in M.RegionalIndices  for S\_t in M.tech\_groups if (S\_t in M.tech\_annual) and ((r, p, S\_t) in M.processVintages.keys())  for S\_v in M.processVintages[r, p, S\_t]  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  max\_act = value(M.MaxActivityGroup[p, g])  expr = activity\_p + activity\_p\_annual <= max\_act  return expr  def MinCapacityGroup\_Constraint(M, p, g):  r"""  Similar to the :code:`MinCapacity` constraint, but works on a group of technologies.  """  min\_cap = value(M.MinCapacityGroup[p, g])  aggcap = sum(  M.V\_CapacityAvailableByPeriodAndTech[r, p, S\_t] \* M.TechGroupWeight[r, S\_t, g]  for r in M.RegionalIndices  for S\_t in M.tech\_groups if (r, p, S\_t) in M.processVintages.keys()  )  expr = aggcap >= min\_cap  return expr  def MaxCapacityGroup\_Constraint(M, p, g):  r"""  Similar to the :code:`MaxCapacity` constraint, but works on a group of technologies.  """  max\_cap = value(M.MaxCapacityGroup[p, g])  aggcap = sum(  M.V\_CapacityAvailableByPeriodAndTech[r, p, S\_t] \* M.TechGroupWeight[r, S\_t, g]  for r in M.RegionalIndices  for S\_t in M.tech\_groups if (r, p, S\_t) in M.processVintages.keys()  )  expr = aggcap <= max\_cap  return expr  def MinInputGroup\_Constraint(M, r, p, i, g):  r"""  Allows users to specify minimum shares of commodity inputs to a group of technologies.  These shares can vary by model time period.  """  inp = sum(  M.V\_FlowOut[r, p, s, d, i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g] / value(M.Efficiency[r, i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v] if S\_i == i  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  inp\_annual = sum(  M.V\_FlowOutAnnual[r, p, i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g] / value(M.Efficiency[r, i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v] if S\_i == i  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  total\_inp = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g] / value(M.Efficiency[r, S\_i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  total\_inp\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g] / value(M.Efficiency[r, S\_i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  min\_inp = value(M.MinInputGroup[r, p, i, g])  expr = (inp + inp\_annual) >= min\_inp \* (total\_inp + total\_inp\_annual)  return expr  def MaxInputGroup\_Constraint(M, r, p, i, g):  r"""  Allows users to specify maximum shares of commodity inputs to a group of technologies.  These shares can vary by model time period.  """  inp = sum(  M.V\_FlowOut[r, p, s, d, i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g] / value(M.Efficiency[r, i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v] if S\_i == i  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  inp\_annual = sum(  M.V\_FlowOutAnnual[r, p, i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g] / value(M.Efficiency[r, i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v] if S\_i == i  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  total\_inp = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g] / value(M.Efficiency[r, S\_i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  total\_inp\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g] / value(M.Efficiency[r, S\_i, S\_t, S\_v, S\_o])  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  max\_inp = value(M.MaxInputGroup[r, p, i, g])  expr = (inp + inp\_annual) <= max\_inp \* (total\_inp + total\_inp\_annual)  return expr  def MinOutputGroup\_Constraint(M, r, p, o, g):  r"""  Allows users to specify minimum shares of commodity outputs to a group of technologies.  These shares can vary by model time period.  """  outp = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, o] \* M.TechGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i] if S\_o == o  for s in M.time\_season  for d in M.time\_of\_day  )  outp\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, o] \* M.TechGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i] if S\_o == o  )  total\_outp = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  total\_outp\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  min\_outp = value(M.MinOutputGroup[r, p, o, g])  expr = (outp + outp\_annual) >= min\_outp \* (total\_outp + total\_outp\_annual)  return expr  def MaxOutputGroup\_Constraint(M, r, p, o, g):  r"""  Allows users to specify maximum shares of commodity outputs to a group of technologies.  These shares can vary by model time period.  """  outp = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, o] \* M.TechGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i] if S\_o == o  for s in M.time\_season  for d in M.time\_of\_day  )  outp\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, o] \* M.TechGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i] if S\_o == o  )  total\_outp = sum(  M.V\_FlowOut[r, p, s, d, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t not in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  for s in M.time\_season  for d in M.time\_of\_day  )  total\_outp\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, S\_t, S\_v, S\_o] \* M.TechGroupWeight[r, S\_t, g]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t in M.tech\_groups  if S\_t in M.tech\_annual  for S\_i in M.processInputs[r, p, S\_t, S\_v]  for S\_o in M.ProcessOutputsByInput[r, p, S\_t, S\_v, S\_i]  )  max\_outp = value(M.MaxOutputGroup[r, p, o, g])  expr = (outp + outp\_annual) <= max\_outp \* (total\_outp + total\_outp\_annual)  return expr |

**LINKED TECHNOLOGIES**

This constraint has been modified with respect to the previous version. In particular, the “LinkedEmissionsTech\_Constraint” definition (from line 2449) has been modified in **temoa\_rules.py**, by substituting with the following.

This is to make the constraint working also with a positive emission activity associated to the primary flow and both with annual and not-annual technologies.

|  |
| --- |
| def LinkedEmissionsTech\_Constraint(M, r, p, s, d, t, v, e):  r"""  This constraint is necessary for carbon capture technologies that produce  CO2 as an emissions commodity, but the CO2 also serves as a physical  input commodity to a downstream process, such as synthetic fuel production.  To accomplish this, a dummy technology is linked to the CO2-producing  technology, converting the emissions activity into a physical commodity  amount as follows:  .. math::  :label: LinkedEmissionsTech  - \sum\_{I, O} \textbf{FO}\_{r, p, s, d, i, t, v, o} \cdot EAC\_{r, e, i, t, v, o}  = \sum\_{I, O} \textbf{FO}\_{r, p, s, d, i, t, v, o}  \forall \{r, p, s, d, t, v, e\} \in \Theta\_{\text{LinkedTechs}}  The relationship between the primary and linked technologies is given  in the :code:`LinkedTechs` table. It is implicit that  the primary region corresponds to the linked technology as well. The lifetimes  of the primary and linked technologies should be specified and identical.  """  linked\_t = M.LinkedTechs[r, t, e]  if (r,t,v) in M.LifetimeProcess.keys() and M.LifetimeProcess[r, linked\_t,v] != M.LifetimeProcess[r, t,v]:  msg = ('the LifetimeProcess values of the primary and linked technologies '  'in the LinkedTechs table have to be specified and identical')  raise Exception( msg )  if (r,t) in M.LifetimeTech.keys() and M.LifetimeTech[r, linked\_t] != M.LifetimeTech[r, t]:  msg = ('the LifetimeTech values of the primary and linked technologies '  'in the LinkedTechs table have to be specified and identical')  raise Exception( msg )  primary\_flow = sum(  M.V\_FlowOut[r, p, s, d, S\_i, t, v, S\_o]\*M.EmissionActivity[r, e, S\_i, t, v, S\_o]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t == t and S\_t not in M.tech\_annual  if S\_v == v  for S\_i in M.processInputs[r, p, t, v]  for S\_o in M.ProcessOutputsByInput[r, p, t, v, S\_i]  )  linked\_flow = sum(  M.V\_FlowOut[r, p, s, d, S\_i, linked\_t, v, S\_o]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t == linked\_t and S\_t not in M.tech\_annual  if S\_v == v  for S\_i in M.processInputs[r, p, linked\_t, v]  for S\_o in M.ProcessOutputsByInput[r, p, linked\_t, v, S\_i]  )  primary\_flow\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, t, v, S\_o]\*M.EmissionActivity[r, e, S\_i, t, v, S\_o]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t == linked\_t and S\_t in M.tech\_annual  if S\_v == v  for S\_i in M.processInputs[r, p, t, v]  for S\_o in M.ProcessOutputsByInput[r, p, t, v, S\_i]  )  linked\_flow\_annual = sum(  M.V\_FlowOutAnnual[r, p, S\_i, linked\_t, v, S\_o]  for S\_r, S\_p, S\_t, S\_v in M.activeActivity\_rptv  if S\_r == r  if S\_p == p  if S\_t == linked\_t and S\_t in M.tech\_annual  if S\_v == v  for S\_i in M.processInputs[r, p, linked\_t, v]  for S\_o in M.ProcessOutputsByInput[r, p, linked\_t, v, S\_i]  )  expr = primary\_flow + primary\_flow\_annual == linked\_flow + linked\_flow\_annual  return expr |

**MAX RESOURCE**

This constraint has been modified with respect to the previous version. In particular, the “MaxResource\_Constraint” definition has been modified in **temoa\_rules.py**, adding the following expression at the end of rows 2110 and 2121.

This is to consider in the computation the different lengths of the time periods.

|  |
| --- |
| \* M.PeriodLength[p] |