Mind Project

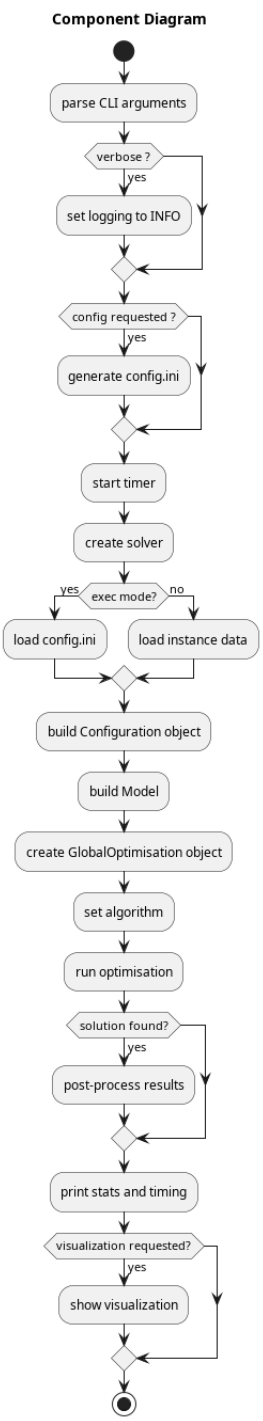
Ana CEBAN

June 4, 2025

Contents

1 Architecture Overview

1.1 Code components and Explanation



Explanation:

1. **Parse arguments:** Command line arguments are parsed (e.g., which instance to run, which solver, algorithm).
   - –**verbose** Enable logging output
   - –**debug** Enable debug mode (solver logging)
   - –**config** Generate default config.ini
   - –**exec** Use *config.ini* for execution, tels the program to load all configuration and instance parameters from the *config.ini* file instead of using command line arguments or defaults
   - –**no**_starting_point Do not generate starting point
   - –**no**_simplified_model Do not use simplified model
   - –**gams** Use GAMS solver
   - –**solver** Specify solver name
   - –**maxiter** Max iterations for algorithms
   - –**maxtime** Time limit for execution
   - –**algorithm** Algorithm choice (multistart, mbh, etc.)
   - –**instance** Use case/instance name
   - –**visualise** Visualize results
   - –**opex** Show opex visualization
   - –**capex** Show capex visualization
   - –**save**_log_sol Save solution log
   - –**version** Print software version
2. **Set Logging** If `-verbose` is set, logging shows more info
3. **Generate Config (if requested)** The *config.ini* file is a dictionnary of algorith/tunning parameters (seeds, epsilon, iteration count) If `-config` is set, a default *config.ini* is generated by the `generate_configuration()` function. The function generates the config based on the default case (default use case : instance specific parameters such as file paths, nr membrane, bounds flags), which is fixed to point to: *Arkema_1.dat Arkema_1_fix_perm.dat Archema_1_eco.dat*

Figure 1: Component Diagram

1. **Start timer** The timer is started to measure execution time.
2. **Solver Setup** If `-exec` is set, the configuration is loaded from *config.ini*, otherwise, the instance is loaded based on the `-instance` argument, and default tuning parameters are used
3. **Build Configuration** The Configuration object is created, it encapsulates all parameters needed for the model construction and optimisation using data from cli arguments, data files or config
4. **Build model** *system.py* The model is built using the configuration and data files (main data, permeability, economics, mask(optional)). It defines all declarative variables, adds all constraints, sets up the objective function for economic cost *obj.py* etc...
5. **Create Solver Instance** A GlobalOptimisation object is created, which will run the optimisation
6. **Set Algorithm** The algorithm is set based on the arguments, (default is multistart). The available algorithm are :
   - **multistart:** Runs the local solver multiple times from different random starting points and keep the best feasible solution
   - **mbh (Monotonic Basin Hopping)** A metaheuristic that perturbs a current solution and performs local search from the perturbed point, accept solution if improved, this repeats over a set number of iterations
   - **global_opt** Runs multistart and then it applies mbh to find a global solution
   - **genetic** Runs a genetic algorithm for optimisation
   - **population** Runs a modified population-based algorithm
7. **Run Optimisation** The selected algorithm is executed
8. **Post-Processing** If a feasible solution is found, post-processing is performed (printing results, saving logs, plotting etc)
9. **Print Statistics and Timing** Prints solver statistic and execution time
10. **Visualisation** Plotting visualisation if requested

1.2 Overview on data files

1.2.1 Main configuration file

- Contains two sections: [tuning] (algorithmic/tuning parameters) and [instance] (file paths, process settings). See Figure 2 for an example.
- When –exec is used, all parameters and file paths are loaded from here
- **Parameters:**
  - **Tunning Section**

    - **pressure_ratio** (float)
    - **epsilon** (dict: keys like At, press_up_f, press_down_f, feed, perm_ref, alpha, delta, xout)
    - **seed1** (int)
    - **seed2** (int)
    - **iteration** (int)
    - **max_no_improve** (int)
    - **max_trials** (int)
    - **pop_size** (int)
    - **generations** (int)
    - **n1_element** (int)
  - **Instance Section**
    - **fname** (str): path to general data file (e.g., Arkema_1.dat)
    - **fname_perm** (str): path to permeability file (e.g., Arkema_1_fix_perm.dat)
    - **fname_eco** (str): path to economics file (e.g., Arkema_1_eco.dat)
    - **num_membranes** (int)
    - **ub_area** (list of float)
    - **lb_area** (list of float)
    - **ub_acell** (list of float)
    - **vp** (bool)
    - **uniform_pup** (bool)
    - **variable_perm** (bool)
    - **fixing_var** (bool)
    - **fname_mask** (str, optional): path to mask file
    - **prototype_data** (str, optional): path to prototype individuals file
    - **log_dir** (str, optional): path to log directory
    - **data_dir** (str, optional): path to data directory

1.2.2 General Data File

- Contains process/system parameters: feed composition, pressure, product purity, bounds, etc...
- Used to define sets, parameters and initial values in the Pyomo model
- **Parameters:**
  - **pressure_in** — Feed pressure
  - **pressure_prod** — (optional) Product pressure
  - **lb_perc_prod** — Lower bound on product purity for each component
  - **ub_perc_prod** — Upper bound on product purity for each component
  - **lb_perc_waste** — Lower bound on waste purity for each component

- **ub_perc_waste** — Upper bound on waste purity for each component
- **normalized_product_qt** — Minimum required product quantity (fraction)
- **final_product** — Name of the final product component
- **FEED** — Feed flow rate (mol/s)
- **XIN** — Feed composition for each component
- **molarmass** — Molar mass for each component
- **ub_press_down** — Upper bound on downstream pressure
- **lb_press_down** — Lower bound on downstream pressure
- **lb_press_up** — Lower bound on upstream pressure
- **ub_press_up** — Upper bound on upstream pressure

  Other possible parameters (from code, not always present):
- **ub_feed**
- **ub_feed_tot**
- **ub_out_prod**
- **ub_out_waste**
- **tol_zero**
- **n** (number of discretization pieces)
- **states** (set of membrane stages)

1.2.3 Permeability File

- Contains membrane type definitions and permeability values for each component
- Used to set up membrane behaviour, assign permeability to each membrane, and define which component is the product/waste
- **Parameters:**
  - **mem_types_set** — Set of membrane types (e.g., ‘A‘)
  - **nb_gas** — Number of gas components
  - **Permeability** — Permeability values for each (membrane type, component) pair
  - **thickness** — Thickness for each membrane type
  - **mem_product** — For each membrane type, which side is the product (‘RET‘ or ‘PERM‘)

    Other possible parameters (from code, not always present):
  - **alpha** (selectivity, if variable permeability is used)
  - **perm_ref** (reference permeability, if variable permeability is used)

1.2.4 Economics File

- Contains economic coefficients and constants (costs, rates, operation time, etc...)
- Used by the objective function (in *obj.py*) to compute costs, CAPEX, OPEX ...
- **Parameters:**
  - **R**— Ideal gas constant
  - **T** — Temperature
  - **eta_cp** — Compressor efficiency
  - **eta_vp_0,** , **eta_vp_1** — Vacuum pump efficiency coefficients
  - **C_cp** — Compressor base cost
  - **C_vp** — Vacuum pump cost factor
  - **C_exp** — Expander base cost
  - **K_m** — Unit cost of membrane module
  - **K_mf** — Base frame cost
  - **K_mr** — Membrane replacement cost
  - **K_el** — Electricity cost factor
  - **K_gp** — Upgraded biogas sales price
  - **K_er** — Exchange rate
  - **MDFc** — Compressor module factor
  - **MPFc** — Material and pressure factor for compressor
  - **MFc** — Module factor for compressor
  - **UF_2000,** 'UF_2011', 'UF_1968' — Update factors
  - **ICF** — Indirect cost factor
  - **nu** — Membrane replacement rate
  - **t_op** — Operation time per year
  - **gamma** — Gas expansion coefficient
  - **phi** — Mechanical efficiency
  - **a** — Annuity coefficient
  - **i** — Interest rate
  - **z** — Project lifetime

    Other possible parameters (from code, not always present): **bp_coef**, **p_coef**, **cmc_coef**, **lti_coef**, **loc_coef**, **dl_coef** (various cost coefficients)

1.2.5 Loading info

The *config.ini* is loaded in the *laucher.py* to provide file paths and parameters, then the general data file and permeability file are loaded in build_model and *system.py* to set up the process and configure the membranes, the economics file is loaded in *obj.py* and used in the ObjFunction to define the cost structure and to use in the objective function. See Figures 3 - 5 for examples of those files.

1.2.6 (Optional: Fixing File)

An additionnal mask file can be used if specified in the *config.ini* file with the option `fixing_var = true` and `fname_mask = ../min/data/fixing/mask.data` for the filepath. When fixing_var is set to true, the mask file is read and used to apply these constraints during model initialization and perturbation. A mask file in the fixing directory usually contains infomation specifying which variables in the optimisation model should be fixed (not changed) and which should free in the applied model. See Figure 6 for an example

Each line has the following syntax:

`fix <variable>:<indexing> <value>`

- fix — indicates this variable is to be fixed.
- <variable> — the name of the model variable (e.g., area, pressure_up, splitFEED_frac).
- <indexing> — the index or indices for the variable (e.g., #1, #2,#3).
- <value> — the value to which the variable should be fixed.

How to interpret :

Example: fix area: #1 5 → the variable area for membrane 1 is fixed to value 5

Parameters that can be fixed via a mask file

- **Membrane-level variables**:
  - **area** - membrane area for each stage
  - **pressure_up** - upstream pressure for each membrane
  - **pressure_down** - downstream pressure for each membrane
  - **Feed_mem**
  - **Flux_RET_mem**
  - **Flux_PERM_mem**
  - **XIN_mem**
  - **X_RET_mem**
  - **X_PERM_mem**
- **Split variables**:
  - **splitFEED** - absolute feed split (not the fraction)
  - **splitFEED_frac** - fraction of retentate from one membrane sent to another
  - **splitRET** #membrane, #membrane - absolute retentate split
  - **splitRET_frac** #membrane, #membrane - fraction of permeate from one membrane sent to another
  - **splitPERM**#membrane, #membrane - absolute permeate split

  - **splitPERM_frac**#membrane, #membrane
  - **splitOutRET** - absolute retentate out
  - **splitOutRET_frac** - fraction of retentate from a membrane sent out of the system
  - **splitOutPERM** - absolute permeate out
  - **splitOutPERM_frac** - fraction of permeate from a membrane sent out of the system

- **System-level variables**:
  - **OUT_prod**
  - **OUT_waste**
  - **XOUT_prod**
  - **XOUT_waste**

- **Membrane type** (if applicable):
  - **mem_type** (sometimes fixed in mask files for architecture) - type of membrane assigned to a stage (e.g., A, B)

2 Solver usage

Knitro is used as the default solver via Pyomo's SolverFactory. It is possible to use any other Pyommo supported solver by launching the program with the `-solver` argument or the config file. Free alternatives supported by Pyomo:

- ipopt (for nonlinear problems, the case here)
- couenne (for MINLP (Mixed-Integer Linear Programming))
- bonmin (MINLP)
- cbc (for MILP/LP (Mixed-Integer Nonlinear Programming))

For our context, ipopt would be a an alternative to try, it is well supported by Pyomo, open source and it handles large-scale nonlinear programming.

```
[tuning]
pressure_ratio = 0.03
epsilon = {'At': 0.3, 'press_up_f': 0.2, 'press_down_f': 0.2,
'feed': 0.3, 'perm_ref': 0.1, 'alpha': 0.1,
'delta': 0.1, 'xout': 0.0001}
seed1 = 2
seed2 = 1
#Nombre d iterations generees par le programme
iteration = 200
max_no_improve = 5
max_trials = 10
pop_size = 30
generations = 5
n1_element = 5

[instance]
data_dir = ../mind/data/
log_dir = ../mind/log/
num_membranes = 2
ub_area = [5 , 5]
lb_area = [0.1, 0.1]
ub_acell = [0.1, 0.1]
# num_membranes = 3
# ub_area = [300 , 300, 300]
# lb_area = [1, 1, 1]
# ub_acell = [0.5, 0.5, 0.5]
fixing_var = False
fname = ../mind/data/CH4_KHIMOD_glob_01.dat
fname_perm = ../mind/data/CH4_KHIMOD_ZEOLITE_perm.dat
fname_eco = ../mind/data/CH4_KHIMOD_ZEOLITE_01_eco.dat
# si fixing_var = false la ligne fname_mask doit etre commentee
# si fixing_var = true le fichier fname_mask doit etre indique
#fname_mask = ../mind/data/fixing/mask.dat
#prototype_data = ../mind/data/prototype_individuals.yaml
uniform_pup = False
vp = false
variable_perm = False
```

Figure 2: Example of config.ini

```
#This file is an example of format for input
  data;

  param pressure_in := 2;#bar
#param pressure_prod := 10;


#set of components of each mixture gaseous
  set components := "H2" "236ea";

#param ub_perc_waste:=
#"N2" 0.001 #0.01,0.005,0.001
#;

  param lb_perc_prod:=
  "236ea" 0.995
  ;

  param normalized_product_qt := 0.95; #0.90,0.95,0.99

  param final_product:= "236ea";

  param FEED := 0.02;#1.483; #mol/s


  param XIN:=
   "H2"  0.829 #0.015
   "236ea" 0.171 #0.985

  ;

  param molarmass:=
    "H2"    2
   "236ea" 152.04
  ;


  param ub_press_down := 1;
  param lb_press_down := 0.01;
  param lb_press_up := 1;
  param ub_press_up := 2;
```

Figure 3: Arkema_1.dat

```
param R := 8.3144659848
param phi := 0.95
param T := 298.15
param gamma := 1.36
param C_cp := 23000
param C_exp := 420
param C_vp := 1000
param eta_cp := 0.85
param K_mr := 25
param K_gp := 0
param K_m := 50
param K_mf := 286000
param K_el := 0.08
param K_er = 0.9
param t_op := 8322
param MFc := 5.11
param nu := 0.25
param UF_2000 := 1.42
param UF_1968 := 4.99
param MPFc := 2.72
param i = 0.08
param z = 15
param eta_vp_1 := 0.1058
param eta_vp_0 := 0.8746
```

Figure 4: Arkema_1_eco.dat

```
# Set of membranes types
set mem_types_set := A

# number of gas (components)
param nb_gas := 2

# permeability values (mem_type component value) A PMP
param Permeability :=
A  H2 130.1
A  236ea  0.1


# thickness
param thickness :=
A := 1


# membrane output
param mem_product :=
A   RET
```

Figure 5: Arkema_1_fix_perm.dat

```
fix area:#1 1644.51
fix area:#2 982.35
fix splitFEED_frac:#1 0
fix splitFEED_frac:#2 1
fix pressure_up:#1 49.08
fix pressure_up:#2 49.08
fix pressure_down:#1 1
fix pressure_down:#2 1
fix splitOutPERM_frac:#1 0
fix splitOutRET_frac:#2 0
fix splitRET_frac:#1,#1 0.000
fix splitRET_frac:#1,#2 0.000
fix splitRET_frac:#2,#1 1.000

fix splitPERM_frac:#1,#1 0.000
fix splitPERM_frac:#1,#2 1.000
```

Figure 6: Example of mask.dat