Exact and Heuristic Methods for the Minimum Number of Matches Problem in Heat Recovery Network Design

This repository contains 1) a collection of benchmark instances of the general heat exchanger network synthesis problem, 2) a source code implementing and evaluating the performance of exact and heuristic methods for the minimum number of matches problem (using Python 2.7.6 and Pyomo 4.4.1), and 3) the obtained results after running the code on an Intel Core i7-4790 CPU 3.60 GHz with 15.6 GB RAM and 64-bit Ubuntu 14.04. Letsios, Kouyialis and Misener (2017) present a technical description of the implemented methods.

Based on the standard sequential method for heat exchanger network design, solving the minimum number of matches problem requires solving the minimum utility cost problem beforehand. In particular, the solution process consists of the following steps:

1. generation of minimum utility cost (i.e. general heat exchanger network design) input files,
2. solving of the minimum utility cost problem using the transportation LP model and generation of the minimum number of matches problem instances,
3. solving of the minimum number of matches problem via exact and heuristic methods.

All input and output data files are located in directory `data`, all required Python modules are located in directory `lib` while the the root file `main.py` performs all steps for obtaining the results.

Benchmark Instances

This repository contains a collection of 48 general heat exchanger network design problem instances. These instances are classified into three test sets:

1. Furman (2000) test set,
2. Chen et al. (2015a, 2015b) test set, and
3. Grossman (2017) test set.

The Furman (2000) instances are manually digitized from the engineering literature, the Chen et al. (2015a, 2015b) are existing instances in the literature, and the Grossman (2017) instances are generated randomly with fixed seeds. Obtaining the Chen et al. (2015a, 2015b) and Grossman (2017) instances requires parsing existing `.gms` files and random selections in the latter case. All minimum utility cost instances as well as the corresponding `.gms` files are stored in the directory `data/original_instances`.

Minimum Utility Cost Problem Instances

An instance of the minimum utility cost problem (class `Min_Utility_Instance` in `lib/problem_classes/min_utility_instance.py`) consists of

- a set of hot streams HS,
- a set of cold streams CS,
- a set of hot utilities HU,
- a set of cold utilities CU, and
- a minimum approach temperature ΔT_{min}.

A hot / cold stream (class `Stream` in `lib/problem_classes/stream.py`) is specified by

- an inlet temperature Tin,
- an outlet temperature Tout, and
- a flow rate heat capacity FCp.

A hot / cold utility (class `Utility` in `lib/problem_classes/utility.py`) is associated with

- an inlet temperature Tin,
- an outlet temperature Tout, and
- a cost κ.

A minimum utility cost problem is instance is stored in `.dat` file under the following format:

```
DTmin 10
HS1 320 200 16.67
HS2 480 280 20
CS1 140 320 14.45
CS2 240 500 11.53
HU1 540 539 0.001
CU1 100 180 0.00005
```

The first line specifies the minimum approach temperature. Every subsequent line and specifies the parameters of either a hot stream (`HS`), cold stream (`CS`), hot utility (`HU`), or cold utility (`CU`) and consists of four elements separated by one or more white spaces. The first element is an identifier indicating whether the line corresponds to a hot stream, cold stream, hot utility or cold utility. The second and third elements correspond to the inlet temperature Tin and outlet temperature Tout, respectively. The fourth element indicates the flow rate heat capacity or the cost depending on whether the line corresponds to a stream or utility, respectively.

Minimum Utility Cost Problem Solving

The code solves an instance of the minimum utility cost problem using the transportation LP model. This model requires a new representation of a problem instance with temperature intervals (class `Transportation_Model` in `lib/problem_classes/transportation_model.py`). The transshipment LP model is solved via CPLEX. Directory `lib/instance_generation` contains the source code for generating the minimum utility cost problem instances and solving them in order to generate the minimum number of matches instances which are stored in directory `data/mip_instances`.

Minimum Number of Matches Problem Instances

An instance of the minimum number of matches problem is represented in the form a transportation network (class `Network` in `lib/problem_classes/network.py`) and is generated after solving an instance of the minimum utility cost problem. A minimum number of matches problem instance does not distinguish between streams and utilities (each utility is considered as a new stream appended to the end of the list of streams). All these instances are stored in directory `data/mip_instances`.

An instance of the minimum number of matches problem consists of the following parameters:

- number of hot streams n,
- number of cold streams m,

- number of temperature intervals k,
- the vectors QH and QC specifying the heat load of each stream in each temperature interval (QH[i,t] and QC[j,t] specify the heat load of hot stream i and cold stream j, respectively, in temperature interval t), and
- the vector R of residual capacities (R[t] is the amount of heat descending from temperature intervals 1,2,...,t on the hot side to temperature intervals t+1,...,k on the cold side).

A minimum number of matches instance is stored in a `.dat` file under the following format:

```
Cost=0.383275
n=3
m=3
k=5
QH[0]: T2 1166.9 T3 833.5
QH[1]: T1 3200.0 T2 800.0
QH[2]: T0 345.9
QC[0]: T1 144.5 T2 1011.5 T3 1445.0
QC[1]: T0 345.9 T1 1844.8 T2 807.1
QC[2]: T4 747.5
R[0]= 0.0
R[1]= 0.0
R[2]= 1210.7
R[3]= 1359.0
R[4]= 747.5
R[5]= 0.0
```

The first line specifies the total utility cost computed by solving the minimum utility cost problem. The second, third and fourth lines specify the number of hot streams, cold streams, and temperature intervals, respectively. For each hot stream i and cold stream j, there is a line starting with `QH[i]:` and `QC[j]:`, respectively, specifying the heat of the stream in each temperature interval. In particular, each stream is associated with a sequence of pairs of the form `T2 1166.9`. Every such pair specifies the heat of the stream in one temperature interval. If a certain temperature interval does not appear in the sequence of stream, this means that the stream has zero heat in the particular temperature interval. Finally, the file contains the heat residual capacities, one for each temperature interval.

Minimum Number of Matches Problem Solving

Exact methods:
CPLEX 12.6.3 and Gurobi 6.5.2 are used to solve exactly both the transshipment and transportation MILP formulations of the minimum number of matches problem. Based on the difficulty of each test set, we set a time limit for each solver run as follows: (i) 1800 seconds for the Furman (2000) test set, (ii) 7200 seconds for the Chen et al. (2015a, 2015b) test set, 14400 seconds for the Grossman (2017) test set. Furthermore, each solver run stores the computed solution together with the following statistics: (i) elapsed time, (ii) number of explored nodes, (iii) upper and lower bound (relative gap). Directory `lib/exact_methods` contains the modules for solving the minimum number of matches problem via exact methods. Directory `data/mip_solutions` stores the computational results which have been obtained via the exact methods.

Heuristic methods:
This code classifies the heuristics for solving the minimum number of matches problem into three

categories: (i) relaxation rounding, (ii) water filling, and (iii) greedy packing. In particular, the problem is solved via the following heuristics:

- Fractional LP Rounding (FLPR), Lagrangian Relaxation Rounding (LRR), Covering Relaxation Rounding (CRR),
- Water Filling Greedy (WFG), Water Filling MILP-based (WFM), and
- Largest Heat Match Greedy (LHM), Largest Heat Match LP based (LHM-LP), Largest Fraction Match (LFM), Shortest Stream (SS).

Directory `lib/heuristic_methods` contains the modules implementing these heuristics. Directory `data/heuristic_solutions` stores the computational results which have been obtained via the heuristic methods.

The experimental results are stored in `.sol` files containing the statistics and the computed solution, i.e. the value of each variable. In the case of exact methods, CPLEX and Gurobi also store their `.log` file.

Repository Contents

- `data`:
  all input and output data (problem instances and obtained results)
  - `original_instances`:
    minimum utility cost problem instances and original `.gms` files
  - `mip_instances`:
    minimum number of matches problem instances obtained after solving the minimum utility cost problem
  - `mip_solutions`:
    computational results obtained via the exact methods
  - `heuristic_solutions`:
    computational results obtained via the heuristic methods
  - `bigM_parameters`:
    big-M parameters for each problem instance computed via the simple standard way, the Gundersen et al. (1997) way and the Letsios et al. (2017) way
- `lib`:
  all required modules for the solution process
  - `instance_generation`:
    generation and solving of minimum utility cost problem instances, generation of minimum number of matches problem instances
  - `problem_classes`:
    essential classes for representing the problem objects
  - `exact_methods`:
    modules implementing the exact methods

    - `heuristic_methods`:
      modules implementing the heuristic methods

    - `io_modules`:
      necessary modules for processing input and output files

- `main.py`:
  root file performing all steps for obtaining the results