A “cut and solve” solver for the single source capacitated facility location problem

Generated by Doxygen 1.8.6

Sun Mar 20 2016 13:08:44

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 generator Class Reference

```
#include <generator.h>
```

Public Member Functions

- generator ()
  Constructor of the data class.
- void generateStidsenAndersenDammannInstance (int n, int m, double BoxLB, double BoxUB, int dLB, int dUB, int sLB, int sUB, int seed)
  Generates an instance of the SSCFLP according to Stidsen, Andersen, and Dammann.
- void generateCornuejolsInstance (int n, int m, int dLB, int dUB, int sLB, int sUB, double Ratio, int seed)
  Generates instance of SSCFLP according to Cornuejols.
- void convertToUFLP ()
- void printToFile (const std::string &FileName)
- int Euclid (std::pair< double, double > p1, std::pair< double, double > p2)
- void clearGenerator ()

Private Attributes

Data

This section contains all the data for describing the SSCFLP.

- int n
- int m
  Number of facility sites.
- int TD
  Number of customers.
- std::vector< std::vector< int > > c
  Total demand. TD=sum_{j=1}^n d[j].
- std::vector< int > f
  Assignment cost. c[i][j] is the cost of assigning customer j to facility i.
- std::vector< int > d
  Fixed opening cost. f[i] is the cost of opening facility i.
- std::vector< int > s
  Demand. d[j] is the demand at customer j.

3.1.1 Constructor & Destructor Documentation

3.1.1.1 generator::generator ( )

Constructor of the data class.

Capacity. s[i] is the capacity of facility i

Constructor of the generator class.

3.1.2 Member Function Documentation

3.1.2.1 void generator::clearGenerator ( )

Clears all data in the generator implying you can use the same generator object to generato several instances.

3.1.2.2 void generator::convertToUFLP ( )

Converts the data to an uncapacitated facility location problem by setting c[i][j] $=$ d[j] $*$ c[i][j], s[i] $=$ m, and d[j] $=$ 1.

3.1.2.3 int generator::Euclid (std::pair< double, double > p1, std::pair< double, double > p2)

Function returning a scaled Euclidean distance between two points (rounded to nearest integer). The distance returned is round (EuclideanDistance (p1, p2) $*$ 10).

Parameters

p1	pair of doubles. Holds the Cartesian coordinates of the first point
p2	pair of doubles. Holds the Cartesian coordinates of the second point

3.1.2.4 void generator::generateCornuejolsInstance (int n, int m, int dLB, int dUB, int sLB, int sUB, double Ratio, int seed)

Generates instance of SSCFLP according to Cornuejols.

Similar to generateStidsenAndersenDammannInstance. Generates coordinates for facilities and customers in unit square. Multipliers distances by 10 and round an use that as unit assignment costs. Capacities and demands are drawn from U(sLB,sUB) and U(dLB,dUB), respectively. Here U(a,b) is a uniform distribution on [a,b]. The capacities are scaled such that total capacity/total demand $=$ Ratio. Finally, the fixed costs are generated as f[i] $=$ U[0,90] $+$ U[100,110]$*$ (s[i]) to reflect economics of scale.

3.1.2.5 void generator::generateStidsenAndersenDammannInstance (int n, int m, double BoxLB, double BoxUB, int dLB, int dUB, int sLB, int sUB, int seed)

Generates an instance of the SSCFLP according to Stidsen, Andersen, and Dammann.

Function generating an instance of the SSCFLP according to the method specified in Stidsen, Andersen, Dammann Management Science (2014), "A Branch and Bound Algorithm for a Class of Biobjective Mixed Integer Programs". The method needs the number of facilities and the number of customers in the instance. For each customer and facility a coordinate in the Euclidean plane is generate in a box [BoxLB , BoxUB]x[Box-LB , BoxUB]. Then the Euclidean distance between each (facility, customer)-pair is calculated and the cost of assigning customer j to facility i is set to c[i][j] $=$ round (dist(i,j) $*$ 10). The fixed costs are set to f[i] $=$ (min_j { c[i][j]+1 }) $*$ 10. Next, the demand of each customer is generated from a uniform distribution on {dLB,...,dUB}. Like wise each facility capacity is generated from {sLB,...,sUB}. Finally a number, r, between 1.5 and 4 is drawn and the capacities are scaled such that $\sum_i s[i]/\sum_j d[j] \approx r$

Parameters

n	integer. Number of facilities.
m	integer. Number of customers.
BoxLB	double. Lower bound on the square box from which coordinates are drawn.
BoxUB	double. Upper bound on the square box from which coordinates are drawn.
dLB	integer. Lower bound on the demands.
dUB	integer. Upper bound on the demands.
sLB	integer. Lower bound on the capacities.
sUB	integer. Upper bound on the capacities.
seed	integer. The seed used to for the random number generator.

3.1.2.6 void generator::printToFile (const std::string & FileName)

Function printing the instance to a file. The format is as follows

```
   n        m
  s[1]     f[1]
  s[2]     f[2]
   ...
  s[n]     f[n]
  d[1]     d[2]     . . .     d[m]
c[1][1]   . . .   c[n][1]
c[2][1]   . . .   c[n][2]
   ...
c[n][1]   . . .   c[n][m]
```

Parameters

FileName	constant reference to string. Contains the name for the file the instance is printed to.

3.1.3 Member Data Documentation

3.1.3.1 std::vector<std::vector<int> > generator::c [private]

Total demand. TD=sum_{j=1}^n d[j].

3.1.3.2 std::vector<int> generator::d [private]

Fixed opening cost. f[i] is the cost of opening facility i.

3.1.3.3 std::vector<int> generator::f [private]

Assignment cost. c[i][j] is the cost of assigning customer j to facility i.

3.1.3.4 int generator::m [private]

Number of facility sites.

3.1.3.5 int generator::n [private]

3.1.3.6 std::vector<int> generator::s [private]

Demand. d[j] is the demand at customer j.

3.1.3.7 int generator::TD [private]

Number of customers.

The documentation for this class was generated from the following files:

- generator.h
- generator.cpp

Chapter 4

File Documentation

4.1 generator.cpp File Reference

```
#include "generator.h"
```

4.2 generator.h File Reference

```
#include <vector>
#include <stdexcept>
#include <iostream>
#include <fstream>
#include <sstream>
#include <limits.h>
#include <random>
#include <chrono>
```

Classes

- class generator

Typedefs

- typedef
  std::uniform_int_distribution UniInt
- typedef
  std::uniform_real_distribution UniReal

  Uniform distribution on integers.
- typedef std::mt19937_64 G
- typedef
  std::chrono::high_resolution_clock myclock

  Random number generator based on the 64-bit Mersenne Twister by Matsumoto and Nishimura, 2000.

4.2.1 Typedef Documentation

4.2.1.1 typedef std::mt19937_64 **G**

4.2.1.2 typedef std::chrono::high_resolution_clock **myclock**

Random number generator based on the 64-bit Mersenne Twister by Matsumoto and Nishimura, 2000.

4.2.1.3 typedef std::uniform_int_distribution **UniInt**

4.2.1.4 typedef std::uniform_real_distribution **UniReal**

Uniform distribution on integers.

4.3 main.cpp File Reference

```
#include <iostream>
#include "generator.h"
```

Functions

- int main ()

4.3.1 Function Documentation

4.3.1.1 int main ( )