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

Generated by Doxygen 1.8.6

Tue Apr 19 2016 14:56:53

Contents

1 An introduction to the SSCFLPsolver class **1**

1.1 License 1

1.2 Description 1

1.3 ILP formulation 2

1.4 Compiling 2

1.5 Example of usage 2

1.5.1 Loading data from a file 2

1.5.2 Loading data from pointers 3

1.5.3 Problematic behaviour 3

1.6 Change log for SSCFLPsolver.h and SSCFLPsolver.cpp 4

2 Class Index **5**

2.1 Class List 5

3 File Index **7**

3.1 File List 7

4 Class Documentation **9**

4.1 rdDat Class Reference 9

4.1.1 Constructor & Destructor Documentation 10

4.1.1.1 rdDat 10

4.1.1.2 ∼rdDat 10

4.1.2 Member Function Documentation 10

4.1.2.1 addFacility 10

4.1.2.2 getAllC 10

4.1.2.3 getAllD 10

4.1.2.4 getAllF 10

4.1.2.5 getAllS 10

4.1.2.6 getC 11

4.1.2.7 getD 12

4.1.2.8 getF 12

4.1.2.9 getNumCust 12

4.1.2.10 getNumFac . . . 12
4.1.2.11 getP . . . 12
4.1.2.12 getS . . . 12
4.1.2.13 getTotalDemand . . . 12
4.1.2.14 rdCFLP . . . 12
4.1.2.15 rdPmed . . . 12
4.1.2.16 rdSSCFLP . . . 13
4.1.2.17 rdUFLP . . . 13
4.1.2.18 setC . . . 13
4.1.3 Member Data Documentation . . . 13
4.1.3.1 c . . . 13
4.1.3.2 d . . . 13
4.1.3.3 f . . . 13
4.1.3.4 m . . . 13
4.1.3.5 n . . . 13
4.1.3.6 p . . . 13
4.1.3.7 s . . . 13
4.1.3.8 TheProblemType . . . 13
4.1.3.9 TotalDemand . . . 13
4.2 solution Class Reference . . . 14
4.2.1 Constructor & Destructor Documentation . . . 14
4.2.1.1 solution . . . 14
4.2.1.2 solution . . . 15
4.2.1.3 ∼solution . . . 15
4.2.2 Member Function Documentation . . . 15
4.2.2.1 getObjVal . . . 15
4.2.2.2 getSolution . . . 15
4.2.2.3 getX . . . 15
4.2.2.4 getY . . . 16
4.2.2.5 setObjVal . . . 16
4.2.2.6 setSolution . . . 16
4.2.3 Member Data Documentation . . . 16
4.2.3.1 m . . . 16
4.2.3.2 n . . . 16
4.2.3.3 ObjVal . . . 16
4.2.3.4 solX . . . 16
4.2.3.5 solY . . . 16
4.3 SSCFLPsolver Class Reference . . . 17
4.3.1 Constructor & Destructor Documentation . . . 20
4.3.1.1 SSCFLPsolver . . . 20

4.3.1.2 ∼SSCFLPsolver . . . 20
4.3.2 Member Function Documentation . . . 21
4.3.2.1 BuildModels . . . 21
4.3.2.2 ChangeDenseToSemiLagrangean . . . 21
4.3.2.3 CheckIfFix . . . 21
4.3.2.4 CleanUp . . . 21
4.3.2.5 CuttingPhase . . . 21
4.3.2.6 fixAfterCuts . . . 21
4.3.2.7 fixBeforeCuts . . . 21
4.3.2.8 getC . . . 23
4.3.2.9 getCapacity . . . 23
4.3.2.10 getCutLower . . . 23
4.3.2.11 getDemand . . . 23
4.3.2.12 getDual . . . 23
4.3.2.13 getF . . . 23
4.3.2.14 getNumCust . . . 24
4.3.2.15 getNumFac . . . 24
4.3.2.16 getObjVal . . . 24
4.3.2.17 getSolution . . . 24
4.3.2.18 getTestStatistics . . . 24
4.3.2.19 getTotalDemand . . . 25
4.3.2.20 greedy . . . 25
4.3.2.21 Load . . . 25
4.3.2.22 Load . . . 25
4.3.2.23 Load . . . 25
4.3.2.24 PrintProgramInfo . . . 26
4.3.2.25 printStats . . . 26
4.3.2.26 RefineAllocation . . . 26
4.3.2.27 RefineLocation . . . 26
4.3.2.28 Run . . . 26
4.3.2.29 RunAsHeuristic . . . 26
4.3.2.30 RunLBHeur . . . 27
4.3.2.31 SepCuts . . . 27
4.3.2.32 setC . . . 27
4.3.2.33 setDualAcentOn . . . 27
4.3.2.34 setF . . . 27
4.3.2.35 setSolution . . . 28
4.3.2.36 solveSparseUsingDualAscent . . . 28
4.3.2.37 twoSwap . . . 29
4.3.3 Member Data Documentation . . . 29

4.3.3.1 AssCst 29
4.3.3.2 avgC 29
4.3.3.3 avgD 29
4.3.3.4 avgF 29
4.3.3.5 avgS 29
4.3.3.6 c 29
4.3.3.7 CnSTime 29
4.3.3.8 CplexOutOff 29
4.3.3.9 CutLowerBound 29
4.3.3.10 CutTime 29
4.3.3.11 d 29
4.3.3.12 debugMe 30
4.3.3.13 DenseCplex 30
4.3.3.14 DenseModel 30
4.3.3.15 DenseObj 30
4.3.3.16 displayStats 30
4.3.3.17 duals 30
4.3.3.18 env 30
4.3.3.19 f 30
4.3.3.20 fixAfterCuttingPhase 30
4.3.3.21 fixBeforeCuttingPhase 30
4.3.3.22 hasCleanedUp 30
4.3.3.23 hasRun 30
4.3.3.24 m 30
4.3.3.25 maxC 30
4.3.3.26 maxD 31
4.3.3.27 maxF 31
4.3.3.28 maxS 31
4.3.3.29 minC 31
4.3.3.30 minD 31
4.3.3.31 minF 31
4.3.3.32 minS 31
4.3.3.33 myEpsilon 31
4.3.3.34 myOne 31
4.3.3.35 myZero 31
4.3.3.36 n 31
4.3.3.37 NumCuts 31
4.3.3.38 ObjVal 32
4.3.3.39 Runtime 32
4.3.3.40 s 32

4.3.3.41 solveSparseUsingDualAscentAlg 32
4.3.3.42 SparseCplex 32
4.3.3.43 SparseModel 32
4.3.3.44 SparseObj 32
4.3.3.45 StartRunTime 32
4.3.3.46 StartTime 32
4.3.3.47 stats 32
4.3.3.48 TD 32
4.3.3.49 x 32
4.3.3.50 xSol 33
4.3.3.51 y 33
4.3.3.52 ySol 33
4.4 testStats Struct Reference 33
4.4.1 Member Data Documentation 34
4.4.1.1 avgNumOfDualItPerSparse 34
4.4.1.2 avgPercentLeft 34
4.4.1.3 bestLowerBound 34
4.4.1.4 bestUpperBound 34
4.4.1.5 CutAndSolveTime 34
4.4.1.6 CuttingTime 34
4.4.1.7 initialUpperBound 34
4.4.1.8 itWhereOptWasFound 34
4.4.1.9 LowerBoundAfterCusts 34
4.4.1.10 m 34
4.4.1.11 n 35
4.4.1.12 numberOfIterations 35
4.4.1.13 numOfSparseSolved 35
4.4.1.14 percentageGap 35
4.4.1.15 time 35
4.4.1.16 WeakLowerBound 35
4.5 TFENCHELopt Struct Reference 35
4.5.1 Member Data Documentation 35
4.5.1.1 Algo 35
4.5.1.2 alpha 35
4.5.1.3 CHK 35
4.5.1.4 Freq 36
4.5.1.5 H 36
4.5.1.6 maxit 36
4.5.1.7 Reduce 36
4.5.1.8 sg_strat 36

4.6 VICcut Struct Reference 36
4.6.1 Detailed Description 36
4.6.2 License 37
4.7 Change log for vico.h 38
4.7.1 Member Data Documentation 38
4.7.1.1 nextcut 38
4.7.1.2 nzcnt 38
4.7.1.3 nzind 39
4.7.1.4 nzval 39
4.7.1.5 rhs 39
4.7.1.6 sense 39
4.7.1.7 UsrDatPtr 39
4.7.1.8 UsrIVal 39
4.7.1.9 UsrRVal 39

5 File Documentation **41**
5.1 main.cpp File Reference 41
5.1.1 Function Documentation 41
5.1.1.1 main 41
5.2 rdDat.cpp File Reference 41
5.3 rdDat.h File Reference 41
5.4 solution.cpp File Reference 42
5.5 solution.h File Reference 42
5.5.1 Detailed Description 42
5.6 SSCFLPsolver.cpp File Reference 42
5.6.1 Function Documentation 43
5.6.1.1 calcPercent 43
5.6.1.2 ILOUSERCUTCALLBACK1 43
5.6.1.3 roundToTwo 43
5.6.2 Variable Documentation 43
5.6.2.1 cutsAddedInCutCallback 43
5.6.2.2 doubleKPcutsAdded 43
5.6.2.3 initializeWithDual1 43
5.6.2.4 initializeWithPercent1 43
5.6.2.5 initializeWithSolution1 43
5.7 SSCFLPsolver.h File Reference 43
5.7.1 Typedef Documentation 44
5.7.1.1 CPUclock 44
5.7.1.2 IloNumMatrix 44
5.7.1.3 IloVarMatrix 44

5.8 vico.h File Reference . . . 44
5.8.1 Macro Definition Documentation . . . 45
5.8.1.1 ALG_CPLEX . . . 45
5.8.1.2 ALG_DP . . . 45
5.8.1.3 ALG_MT1 . . . 45
5.8.1.4 SG_DEFAULT . . . 45
5.8.1.5 SG_DEFLECT . . . 45
5.8.1.6 SG_SMOOTH . . . 45
5.8.2 Typedef Documentation . . . 45
5.8.2.1 TFENCHELopt . . . 45
5.8.2.2 VICcut . . . 45
5.8.3 License . . . 46
5.9 Change log for vico.h . . . 47
5.9.1 Function Documentation . . . 47
5.9.1.1 VICaddtolst . . . 47
5.9.1.2 VICallocut . . . 48
5.9.1.3 VICcflfc . . . 48
5.9.1.4 VICcflsmi . . . 49
5.9.1.5 VICclearlst . . . 49
5.9.1.6 VICeci . . . 50
5.9.1.7 VICecikl . . . 50
5.9.1.8 VICfreecut . . . 50
5.9.1.9 VICfreelst . . . 50
5.9.1.10 VICgapfn . . . 51
5.9.1.11 VICisintvec . . . 51
5.9.1.12 VICkconf . . . 52
5.9.1.13 VICkpreduce . . . 52
5.9.1.14 VICkpsep . . . 53
5.9.1.15 VIClci . . . 54
5.9.1.16 VICsearchcut . . . 54
5.9.1.17 VICsetdefaults . . . 55
5.9.1.18 VICsort . . . 55
5.9.1.19 VICuflcov . . . 55
5.9.1.20 VICuflohi . . . 56

Chapter 1

An introduction to the SSCFLPsolver class

Author

Sune Lauth Gadegaard

Version

1.2.0

1.1 License

If you use the software in any academic work, please make a reference to

S.L. Gadegaard and A. Klose and L.R. Nielsen, (2015), "An exact algorithm based on cutting planes and local branching for the single source capacitated facility location problem", Technical report, CORAL, Aarhus University, sgadegaard@econ.au.dk.

1.2 Description

Class implementing a solver for the single sourced capacitated facility location problem (SSCFLP) The algorithm implememnted here is presented in Gadegaard et al. (2015) "An exact algorithm based on cutting planes and local branching for the single source capacitated facility location problem". The idea of the algorithm is based on the cut–and–solve framweork presented in S. Climer and W. Zhang, (2016), "Cut-and-solve: An iterative search strategy for combinatorial optimization problems", Artificial Intelligence, 170:714-738. We iterate between solving a constrianed relaxation (the Dense problem) of SSCFLP and a restricted version of SSCFLP (the Sparse problem). This leads to a framework where a constrained CFLP is used as the Dense problem and a smaller SSCFLP is used for the Sparse problem.

1.3 ILP formulation

Given a set of potentil facility sites, I, and a set of customers, J, the SSCFLP can be formulated as follows: Let $f_i > 0$ be the cost of opening facility i, and c_{ij} the cost of assigning customer j to facility i. Furthermore, let $d_j > 0$ the demand for a certain good at customer j and $s_i > 0$ the capacity for the good at facility i. Introducing binary variable y_i which is equal to one if and only if a facility is open at facility i and binary variable x_{ij} indicating if customer j is assigned to facility i ($x_{ij} = 1$) or not ($x_{ij} = 0$), the SSCFLP can be formulated as the linear integer programming problem

$$\min \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} + \sum_{i\in I} f_i y_i \tag{O}$$

$$\text{s.t.: } \sum_{i\in I} x_{ij} = 1, \quad \forall j \in J, \tag{A}$$

$$\sum_{j\in J} d_j x_{ij} \le s_i y_i, \quad \forall i \in I, \tag{C}$$

$$x_{ij} - y_i \le 0, \quad \forall i \in I,\ j \in J. \tag{GUB}$$

$$\sum_{i\in I} s_i y_i \ge D = \sum_{j\in J}, \tag{TD}$$

$$x_{ij},\ y_i \in \{0,1\}, \quad \forall i \in I,\ j \in J. \tag{B}$$

Here (O) minimizes the total cost, composed of assignment costs and fixed opening costs. Constraints (A) ensure that all customers are assigned to exactly one facility while the constraints (C) make sure that the no customer is assined to a closed facility and that each facility's capacity is respected. The generalized upper bounds (GUB) are infact redundant (implied by (C)), but the improve the LP relaxation quite a lot. The total demand constraint (TD) is likewise redundant, but its structure is used in the cutting plane part of the algorithm implemented here. Lastly, (B) require all variables to be binary.

1.4 Compiling

The codes were compiled using the GNU GCC compiler on a Linux Ubuntu 14.04 machine. The following flags were used: -Wall -O3 -std=c++11 -DIL_STD. The Code::blocks IDE was used as well. Information on how to configure Code::Blocks IDE with CPLEX on a Linux machin can be found here: http://www-01.ibm.com/support/docview.wss?uid=swg21449771

1.5 Example of usage

This section contains two examples of how the SSCFLPsolver can be used. The first example loads data from a file provided as the first argument to the main function while the other loads data into the SSCFLPsolver object using pointers to arrays.

1.5.1 Loading data from a file

This is a simple example using a datafile provided as an argument to the main function

```
// main.cpp
#include"SSCFLPsolver.h"
int main(int argc, char** argv){
    try{
        SSCFLPsolver solver = SSCFLPsolver();
        // the second argument specifies the format of the data file.
        solver.Load(argv[1],1);
        if(solver.Run()){
            std::cout << "Hurra!" << std::endl;
        }else{
```

```
                std::cout << "Bummer!" << std::endl;
            }
            return 0;
        }catch(std::exception &e){
            std::cerr << "Exception: " << e.what() << std::endl;
            exit(1);
        }catch(...){
              std::cerr << "An unexpected exception was thrown. Caught in main.\n";
              exit(1);
        }
    }
```

1.5.2 Loading data from pointers

In this example we load the data using a user-provided function called `GetMyData`.

```
    //main.cpp
    #include"SSCFLPsolver.h"
    int main(){
        try{
            int n, m;
            int* f, s, d;
            int** c;
            GetMyData((n, m, c, f, d, s);
            SSCFLPsolver solver = SSCFLPsolver();
            solver.Load(n, m, c, f, d, s);
            if(solver.Run()){
            std::cout << "Hurra!" << std::endl;
            }else{
                std::cout << "Bummer!" << std::endl;
            }
        }catch(std::exception &e){
            std::cerr << "Exception: " << e.what() << std::endl;
            exit(1);
        }catch(...){
            std::cerr << "An unexpected exception was thrown. Caught in main.\n";
            exit(1);
        }
    }
```

1.5.3 Problematic behaviour

One kind of not so brilliant thing about this implementation is that you cannot call the run function multiple times, as an IloConversion is used after the cutting plane phase. The IloConversion can only be called once per model! A possible workaround would be to simply have a pure cutting plane IloModel internally in the class which could be dedicated solely to the generation of cutting planes.

1.6 Change log for SSCFLPsolver.h and SSCFLPsolver.cpp

FILE:	SSCLPsolver.h and SSCFLPsolver.cpp		
Version:	1.2.0		
- -			
CHANGE LOG:	DATE	VER.-NO.	CHANGES MADE
	2015–03–01	1.0.0	First implementation
	2015–04–09	1.1.0	Cutting planes added
	2015–04–13	1.1.1	Variable fixation in the sparse problem added.
	2015–04–14	1.1.1	A number of get/set functions and a "setSolution" function which sets a solution internally in cplex was added.
	2015–04–01	1.1.2	Added exact separation from the effective capacity polytope with generalized upper bounds. Does not seem to increase the bound that much
	2016–04–19	1.2.0	Added functionality to solve the sparse problems using dual ascent. Works best when ratio between total capacity and total demand ins small (≤ 3)

Chapter 2

Class Index

2.1 Class List

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

rdDat . . . 9
solution . . . 14
SSCFLPsolver . . . 17
testStats . . . 33
TFENCHELopt . . . 35
VICcut
Implements seperations routines several different problems . . . 36

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

main.cpp 41
rdDat.cpp 41
rdDat.h 41
solution.cpp 42
solution.h 42
SSCFLPsolver.cpp 42
SSCFLPsolver.h 43
vico.h 44

Chapter 4

Class Documentation

4.1 rdDat Class Reference

`#include <rdDat.h>`

Public Member Functions

- rdDat (std::string Filename, int ProblemType)
- ~rdDat ()
- void rdPmed (std::string DataFile)
- void rdUFLP (std::string DataFile)
- void rdCFLP (std::string DataFile)
- void rdSSCFLP (std::string DataFile)
- int getNumFac ()
- int getNumCust ()
- int getP ()
- int getD (int j)
- int getS (int i)
- int getF (int i)
- int getC (int i, int j)
- std::vector< int > getAllD ()
- std::vector< int > getAllS ()
- std::vector< double > getAllF ()
- std::vector< std::vector
  < double > > getAllC ()
- void addFacility (int cap, int fixed, std::vector< double > newC)
- void setC (int i, int j, int cost)
- int getTotalDemand ()

Private Attributes

- int n

  Number of facilities.
- int m

  Number of customers.
- int p

  Number of open faiclities in a solution to the p-median problem.
- int TotalDemand

Sum of the demands.

- std::vector< int > d

  Demands. d[j] demand of customer.
- std::vector< int > s

  Capacities. s[i] capacity of facility i.
- std::vector< double > f

  Fixed opening cost. f[i] cost of opening facility i.
- std::vector< std::vector
  < double > > c

  Assingment cost. c[i][j] is the cost of supplying all of customer j's demand from facility i.
- int TheProblemType

  Integer indicating which problem type is in qustion.

4.1.1 Constructor & Destructor Documentation

4.1.1.1 rdDat::rdDat (std::string Filename, int ProblemType)

Constructor of the rdDat class.

Parameters

Filename	String. Contains the path to a data file of appropriate format

4.1.1.2 rdDat::∼rdDat ( )

Destructor of the class. Cleans up after the us.

4.1.2 Member Function Documentation

4.1.2.1 void rdDat::addFacility (int cap, int fixed, std::vector< double > newC)

! Adds an extra facility to the data

Parameters

cap	constant re

4.1.2.2 std::vector<std::vector<double> > rdDat::getAllC ( ) `[inline]`

Returns a pointer to a pointer to the an integer array containing the assignment costs

4.1.2.3 std::vector<int> rdDat::getAllD ( ) `[inline]`

Returns a pointer to the first element in the integer array containing the demands

4.1.2.4 std::vector<double> rdDat::getAllF ( ) `[inline]`

Returns a pointer to the first element in the integer array containing the fixed opening costs

4.1.2.5 std::vector<int> rdDat::getAllS ( ) `[inline]`

Returns a pointer to the first element in the integer array containing the capacities

4.1.2.6 int rdDat::getC (int i, int j) [inline]

Returns the assignment cost of the the facility–customer pair (i,j)

Parameters

i	integer. Index of the facility
j	integer. Index of the customer

4.1.2.7 int rdDat::getD (int j) [inline]

Returns the demand of customer j

Parameters

j	integer. Index of the customer who's demand you want

4.1.2.8 int rdDat::getF (int i) [inline]

Returns the fixed opening cost of facility i

Parameters

ji	integer. Index of the facility who's fixed cost you want

4.1.2.9 int rdDat::getNumCust ( ) [inline]

Returns the number of customers

4.1.2.10 int rdDat::getNumFac ( ) [inline]

Returns the number of facilities

4.1.2.11 int rdDat::getP ( ) [inline]

Returns the number facilities which must be open in a p-median problem

4.1.2.12 int rdDat::getS (int i) [inline]

Returns the capacity of facility i

Parameters

i	integer. Index of the facility who's capacity you want

4.1.2.13 int rdDat::getTotalDemand ( ) [inline]

Returns the total demand

4.1.2.14 void rdDat::rdCFLP (std::string DataFile)

Reads the data of a capacitated facility location problem problem

4.1.2.15 void rdDat::rdPmed (std::string DataFile)

Reads the data of a p-median problem

4.1.2.16 void rdDat::rdSSCFLP (std::string DataFile)

Reads the data of a single source capacitated facility location problem problem

4.1.2.17 void rdDat::rdUFLP (std::string DataFile)

Reads the data of an uncapacitated facility location problem problem

4.1.2.18 void rdDat::setC (int i, int j, int cost) [inline]

! Set the assignment cost c[i][j]

4.1.3 Member Data Documentation

4.1.3.1 std::vector<std::vector<double> > rdDat::c [private]

Assingment cost. c[i][j] is the cost of supplying all of customer j's demand from facility i.

4.1.3.2 std::vector<int> rdDat::d [private]

Demands. d[j] demand of customer.

4.1.3.3 std::vector<double> rdDat::f [private]

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

4.1.3.4 int rdDat::m [private]

Number of customers.

4.1.3.5 int rdDat::n [private]

Number of facilities.

4.1.3.6 int rdDat::p [private]

Number of open faiclities in a solution to the p-median problem.

4.1.3.7 std::vector<int> rdDat::s [private]

Capacities. s[i] capacity of facility i.

4.1.3.8 int rdDat::TheProblemType [private]

Integer indicating which problem type is in qustion.

4.1.3.9 int rdDat::TotalDemand [private]

Sum of the demands.

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

- rdDat.h
- rdDat.cpp

4.2 solution Class Reference

`#include <solution.h>`

Public Member Functions

- solution (int nn, int mm)
  Overloaded constructor of the solution class.
- solution (int nn, int mm, int ∗newY, int ∗newX, int obj)
  Overloaded constructor of the solution class.
- ∼solution ()
  Destructor of the solution class.
- void setSolution (int ∗newY, int ∗newX, int obj)
  Set the solution.
- void getSolution (int ∗getY, int ∗getX)
  Returns the solution to the y-variables.
- void setObjVal (int &obj)
  Sets the objective function value of the solution.
- int getY (int i)
  Gives the solution value of solY[i].
- int getX (int j)
  Gives the solution value of solX[j].
- int getObjVal ()
  Returns the objective function value of the solution.

Private Attributes

- int ObjVal
  Objective function value of the solutuion.
- int ∗ solY
  Pointer to array of integers. solY[i] = 1 if facility i is open in the solution. Otherwise solY[i]=0.
- int ∗ solX
  Pointer to array of integers. solX[j] = i if and only customer j is assingned to facility i in the solution.
- int n
  Integer. The number of facilities in the instance of the SSCFLP for which this is a solution.
- int m
  Integer. The number of customers in the instance of the SSCFLP for which this is a solution.

4.2.1 Constructor & Destructor Documentation

4.2.1.1 solution::solution (int nn, int mm)

Overloaded constructor of the solution class.

Constructor of the solution class. Initializes the number of facilities and the number of customers. Allocattes memory for the solY and the solX arrays. Initializes arrays to zero, that is solY[i]=solX[j]=0 for all i and j

Parameters

nn	Integer. The number of facilities.
mm	Integer. The number of customers.

4.2.1.2 solution::solution (int nn, int mm, int ∗ newY, int ∗ newX, int obj)

Overloaded constructor of the solution class.

Constructor of the solution class. Initializes the number of facilities and the number of customers. Allocattes memory for the solY and the solX arrays. Sets the value of solY[i]=newY[i] and solX[j]=newX[j]. If newY==NULL or newX==NUL ObjVal is set to -1 indicating the solution is rubbish.

Parameters

nn	Integer. The number of facilities.
mm	Integer. The number of customers.
newY	Pointer to array of integers. Contains the y-values for the solution you want to store. newY[i]=1 iff facility i is open.Ŕ
newX	Pointer to array of integers. Contains the assignment of the solution you want to store. newX[j]=i iff customer j is assingned to facility i.
obj	Integer. The objective function value of the soltuion.

4.2.1.3 solution::∼solution ( )

Destructor of the solution class.

Destructor of the solution class. Clears all allocated memory.

4.2.2 Member Function Documentation

4.2.2.1 int solution::getObjVal ( ) [inline]

Returns the objective function value of the solution.

Returns the objective function value of the solution. Remember if ObjVal=-1 it means the solution is rubbish!

4.2.2.2 void solution::getSolution (int ∗ getY, int ∗ getX)

Returns the solution to the y-variables.

Returns the solution to the y-variables stored in the object. If setSolution() or solution(int,int,<em>int,∗int) has not been called, getSolution(int,int∗) will return rubbish.

Parameters

getY	pointer to integer array of size at least n. Contains on output the solution to the y variables stored in the solution object. getY[i]=1 iff faiclity i is open in the solution.
getX	pointer to integer array of size at least m. Contains on output the solution to the assignments stored in the solution object. getX[j]=i iff customer j is assinged to facility i in the solution.

4.2.2.3 int solution::getX (int j)

Gives the solution value of solX[j].

Gives the solution value of solX[j]. If j>=m or j<0 the last or first element will be returned, respectively. No error is thrown, but a message is displayed.

4.2.2.4 int solution::getY (int i)

Gives the solution value of solY[i].

Gives the solution value of solY[i]. If i>=n or i<0 the last or first element will be returned, respectively. No error is thrown, but a message is displayed.

4.2.2.5 void solution::setObjVal (int & obj) [inline]

Sets the objective function value of the solution.

4.2.2.6 void solution::setSolution (int * newY, int * newX, int obj)

Set the solution.

Sets the solution equal to the new solution provided by (newY, newX). If either newY = NULL or newX = NULL ObjVal is set to -1 to indicate no valid solution is present.

Parameters

newY	pointer to array of integers. Must be of size at least n. newY[i]=1 iff facility i is open in the solution you want to store.
newX	pointer to array of integers. Must be of size at least m. newX[j]=i iff customer j is assigned to faiclity i in the solution you want to store.
obj	Integer. The objective function value of the solution provided.

4.2.3 Member Data Documentation

4.2.3.1 int solution::m [private]

Integer. The number of customers in the instance of the SSCFLP for which this is a solution.

4.2.3.2 int solution::n [private]

Integer. The number of facilities in the instance of the SSCFLP for which this is a solution.

4.2.3.3 int solution::ObjVal [private]

Objective function value of the solutuion.

4.2.3.4 int* solution::solX [private]

Pointer to array of integers. solX[j] = i if and only customer j is assingned to facility i in the solution.

4.2.3.5 int* solution::solY [private]

Pointer to array of integers. solY[i] = 1 if facility i is open in the solution. Otherwise solY[i]=0.

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

- solution.h
- solution.cpp

4.3 SSCFLPsolver Class Reference

```
#include <SSCFLPsolver.h>
```

Public Member Functions

- SSCFLPsolver ()
  The constructor of the class.
- ∼SSCFLPsolver ()
  The destructor of the class.
- void Load (char ∗FileName, int format)
  Public function for loading data.
- void Load (int nn, int mm, int ∗∗cc, int ∗ff, int ∗dd, int ∗ss)
  Public function for loading data.
- void Load (rdDat ∗data)
  Public function for loading the data of a SSCFLP.
- bool setSolution (int nn, int mm, int ∗yval, int ∗xass, int UBval)
  Hands a solution to the solver.
- int getObjVal ()
  Returns the optimal objective function value.
- void printStats ()
  Displays statistics for an instance of SSCFLP loaded into the SSCFLPsolver object.
- bool Run ()
  This is the main function in the SSCFLPsolver class.
- bool RunAsHeuristic ()
  Runs a local branching heuristic.
- void getSolution (int ∗getY, int ∗getX)
  Method for retrieving an optimal solution to the instance of SSCFLP.
- int getC (int i, int j)
  Returns the assignment cost c[i][j].
- int getF (int i)
  Returns the fixed opening cost f[i].
- void setC (int i, int j, int cc)
  Sets the assignment cost c[i][j].
- void setF (int i, int ff)
  Sets the fixed opening cost f[i].
- int getNumFac ()
  Returns the number of facilities.
- int getNumCust ()
  Returns the number of customers.
- IloNum getDual (int j)
- int getDemand (int indx)
- int getCapacity (int indx)
- int getTotalDemand ()
- double getCutLower ()
- void fixBeforeCuts (int i, int j)
  Fixes the x[i][j] variable to the zero before adding cuts.
- void fixAfterCuts (int i, int j)
  Fixes the x[i][j] variable to the zero after adding cuts.
- testStats ∗ getTestStatistics ()
  Returns the test statistics gathered during the optimization.
- void setDualAcentOn ()
  Enables the dual ascent algorithm.

Public Attributes

Public variables

This section contains all publicly accesible variables

- IloNumVarArray y
  IloNumVarArray used for the location variables.
- IloVarMatrix x

Private Member Functions

- void BuildModels ()
  Builds the IloModels.
- void PrintProgramInfo ()
  Prints info about the program and the author.
- bool SepCuts ()
  Separates an LP solution (x,y) from the capacity constraints.
- void CuttingPhase ()
  Runs the cutting plane phase.
- void CheckIfFix (std::vector< int > &ones)
  Checks if all y-variables can be fixed in the sparse problem.
- void ChangeDenseToSemiLagrangean ()
- void RunLBHeur ()
  Runs a simple local branching heuristic.
- void RefineLocation ()
  Performs local branching on the locational variables.
- void RefineAllocation ()
  Performs local branching on the allocation varibales.
- void greedy ()
  A greedy algorithm for SSCFLP.
- bool solveSparseUsingDualAscent (double &NewObjective, solution &incumbent)
  A dual ascent algorithm for the sparse problems.
- void twoSwap ()
- void CleanUp ()
  Cleans up the memory.

Private Attributes

- testStats * stats
  Pointer to an instance of the testStats struct. Contains on termination the statistics gathered during optimization.

Cplex

This section contains all the cplex gear needed for the algorithm to run.

- IloEnv env
  The environment used throughout the lifetime of the object.
- IloModel SparseModel
  The IloModel used for the sparse problem.
- IloModel DenseModel
  The IloModel used for the dense problem.
- IloCplex SparseCplex
  The IloCplex environment used for the sparse problem.

- IloCplex DenseCplex
  The IloCplex environment used for the dense problem.
- IloObjective DenseObj
  Extractable holding the objective function of the dense problem.
- IloObjective SparseObj
  Extractable holding the objective function of the sparse problem.
- IloRangeArray AssCst
  Holds the assignemnt constraints of the denseproblem only!
- IloNumArray duals
  Holds the duals of the assingment constraints of the dense problem.
- std::vector< std::pair< int, int > > fixAfterCuttingPhase
- std::vector< std::pair< int, int > > fixBeforeCuttingPhase

Data

This section contains all the data for describing the SSCFLP.

- IloInt n
  Number of facility sites.
- IloInt m
  Number of customers.
- IloNumMatrix c
  Assignment costs. c[i][j] = cost of assigning customer j to facility i.
- IloNumArray f
  Fixed opening costs. f[i] = cost of opening facility i.
- IloNumArray d
  Demands. d[j] = demand at customer j.
- IloNumArray s
  Capacities; s[i] = capacity at customer i.
- IloInt TD

Parameters and flags

This section contains a list of parameters and flags used internally in the SSCFLPsolver class.

- bool hasCleanedUp
  Used to flag if the CleanUp function has been called.
- bool hasRun
  Used to flag if the Run() function has been called.
- bool displayStats
  Used to flag if statistics should be printed to screen.
- bool CplexOutOff
  Used to flag that cplex' output should be redirected to the null stream.
- bool debugMe
  Used to flag if debug print outs should be enabled.
- bool solveSparseUsingDualAscentAlg
  True if one should use dual ascent for solving the sparse problems. Defaults to false.
- int ObjVal
  After the Run() has been called, ObjVal contains the optimal solution value.
- int maxF
  Contains the maximum value for the fixed opening costs after calling Load().
- int minF
  Contains the minimum value for the fixed opening costs after calling Load().
- int maxC
  Contains the maximum value for the assignment costs after calling Load().
- int minC
  Contains the minimum value for the assignment costs after calling Load().
- int minD
  Contains the minimum value for the customer demands after calling Load().

- int maxD
  Contains the maximum value for the customer demands after calling Load().
- int minS
  Contains the minimum value for the facility capacities after calling Load().
- int maxS
  Contains the maximum value for the facility capacities after calling Load().
- double avgF
  Contains the average value of the fixed costs after calling Load().
- double avgC
  Contains the average value of the assignment costs after calling Load().
- double avgD
  Contains the average value of the customer demands after calling Load().
- double avgS
  Contains the average value of the facility capacities after calling Load().
- double StartTime
  Clock time where the SSCFLPsolver object is initialized.
- double StartRunTime
  Clock time where the Run() function is called.
- double CutTime
  CPU seconds used in the cutting phase.
- double CnSTime
  CPU seconds used in the cut and solve phase.
- double Runtime
  CPU seconds used for the whole procedure.
- double CutLowerBound
  The lower bound produced by the cutting plane algorithm.
- int ∗ ySol
  Pointer to an array of integers. If ySol[i]=1 facility i is open.
- int ∗ xSol
  Pointer to an array of integers. If xSol[j]=i customer j is assigned to facility i.
- int ∗ NumCuts

Tolerances

Tolerances used to control floating point arithmetic. All values are initialized in the constructor of the class.

- double myZero
  My interpretation of "zero". Constructor initialized.
- double myOne
  My interpretation of "one". Constructor initialized.
- double myEpsilon

4.3.1 Constructor & Destructor Documentation

4.3.1.1 SSCFLPsolver::SSCFLPsolver ( )

The constructor of the class.

The constructor of the class SSCFLPsolver. Initializes the data containers, IloModels, and IloCplex environments.

4.3.1.2 SSCFLPsolver::∼SSCFLPsolver ( )

The destructor of the class.

The destructor of the class SSCFLPsolver. Frees all memory allocated during the lifetime of the class object.

4.3.2 Member Function Documentation

4.3.2.1 void SSCFLPsolver::BuildModels ( ) [private]

Builds the IloModels.

This function builds the model on the IloModels defined above and exports them to approproate cplex environments.

4.3.2.2 void SSCFLPsolver::ChangeDenseToSemiLagrangean ( ) [private]

Change the dense problem to a semi lagrangean of the SSCFLP. As Lagrangean dual multiplier, we use the largest dual cost of the assignemnt constraints of the cut–enhanced SSCFLP

4.3.2.3 void SSCFLPsolver::CheckIfFix (std::vector< int > & ones) [private]

Checks if all y-variables can be fixed in the sparse problem.

Checks if all y-variables can be fixed in the sparse problem. Uses the combinatorial argument that if $sum_{i: 0<=y_i<=1}s_i -s_{{i}}<=D$ for some D, then y_{i} can be fixed to one.

Parameters

ones	std::vector of integers. Contains the free variables on input and the variables which can be fixed to one on output.

4.3.2.4 void SSCFLPsolver::CleanUp ( ) [private]

Cleans up the memory.

This function is used to clean up the class SSCFLPsolver in case of exceptions or other unpleasanties. Also used as the base function in the destructor.

4.3.2.5 void SSCFLPsolver::CuttingPhase ( ) [private]

Runs the cutting plane phase.

Runs the cutting plane phase corresponding to Phase 1, in S.L. Gadegaard and A. Klose and L.R. Nielsen, (2015), “An exact algorithm based on cutting planes and local branching for the single source capacitated facility location problem”, Technical report, CORAL, Aarhus University, sgadegaard@econ.au.dk.

4.3.2.6 void SSCFLPsolver::fixAfterCuts (int i, int j) [inline]

Fixes the x[i][j] variable to the zero after adding cuts.

Fixes the x[i][j] variable to the zero after adding cuts. This means that if a x[i][j] should be fixed to one, then all other assignments must be set to zero for the corresponding customer.

Parameters

i	integer. Index of the facility.
j	integer. Index of the customer.

4.3.2.7 void SSCFLPsolver::fixBeforeCuts (int i, int j) [inline]

Fixes the x[i][j] variable to the zero before adding cuts.

Fixes the x[i][j] variable to the zero before adding cuts. This means that if a x[i][j] should be fixed to one, then all other assignments must be set to zero for the corresponding customer.

Parameters

i	integer. Index of the facility.
j	integer. Index of the customer.

4.3.2.8 int SSCFLPsolver::getC (int i, int j) [inline]

Returns the assignment cost c[i][j].

Method returning the cost of assigning customer j to facility i

Parameters

i	integer. Index of the facility you want.
j	integer. Index of the customer you want.

Note

The index i has to be less than getNumFac() and j has to be less than getNumCust(). Both i and j need to be non-negative. No exceptions are thrown if you fuck it up!

Returns

The cost of assigning customer j to facility i

4.3.2.9 int SSCFLPsolver::getCapacity (int indx) [inline]

Returns the capacity of the facility specified by the index

Parameters

indx	integer. Index of the facility for which you want the capacity

4.3.2.10 double SSCFLPsolver::getCutLower ( ) [inline]

Returns the lower bound obtained by the cutting plane algorithm

4.3.2.11 int SSCFLPsolver::getDemand (int indx) [inline]

Returns the demand of cutomer specified by index

Parameters

indx	integer. Index of the customer for which you want the demand

4.3.2.12 IloNum SSCFLPsolver::getDual (int j) [inline]

Returns the optimal value of the dual variable of the specified assignment constraint. The duals are of the cut-enhanced problem

4.3.2.13 int SSCFLPsolver::getF (int i) [inline]

Returns the fixed opening cost f[i].

Parameters

i	integer. The index of the facility who's fixed cost you want.

Note

i>=0 and i<getNumFac()! No exceptions are thrown if you fuck it up!

Returns

The fixed incurred while opening facility i

4.3.2.14 int SSCFLPsolver::getNumCust ( ) [inline]

Returns the number of customers.

Returns

The number of customers.

4.3.2.15 int SSCFLPsolver::getNumFac ( ) [inline]

Returns the number of facilities.

Returns

The number of facility sites.

4.3.2.16 int SSCFLPsolver::getObjVal ( ) [inline]

Returns the optimal objective function value.

Returns the optimal objective function value. Can only be called after the Run() function has been called. An exception is thrown if getObjVal() is called prior to Run()

4.3.2.17 void SSCFLPsolver::getSolution (int * getY, int * getX)

Method for retrieving an optimal solution to the instance of SSCFLP.

Method for retrieving an optimal solution to the instance of SSCFLP. Note that one needs to call Run() before calling getSolution as there will be no solution otherwise.

Parameters

getY	Pointer to integer array of size at least n. Contains on output an optimal solution to the y-variables. getY[i]=1 if facility i is open in an optimal solution. If for some reason no solution is stored in the class, getY = 0 on output, that is the null-pointer is returned.
getX	Pointer to integer array of size at least m. Contains on output an optimal solution to the x-variables. getX[j]=i if customer j is assigned to facility i in an optimal solution. If for some reason no solution is stored in the class, getX = 0 on output, that is the null-pointer is returned.

4.3.2.18 **testStats*** SSCFLPsolver::getTestStatistics ( ) [inline]

Returns the test statistics gathered during the optimization.

Returns the test statistics as a pointer to the struct-type testStats.

4.3.2.19 int SSCFLPsolver::getTotalDemand ( ) [inline]

Returns the total demand of the instances. Thas is sum(j) d(j).

4.3.2.20 void SSCFLPsolver::greedy ( ) [private]

A greedy algorithm for SSCFLP.

Algorithm used if no feasible solution could be found. The method tries in a greedy way to assign the customer with largest demand to facility with largest residual capacity. The method usually finds a (low quality) solution! The method starts by solving the binary knapsack problem max{ sum(i) f(i)∗z(i) : sum(i) s(i)∗z(i)<= sum(i)s(i)-sum(j)d(j)} in order to find an initial set of open facilities. An facility is initially open if z(i)=0 in an optimal solution to the knapsack problem. Then the set of customers is sorted in non–increasing order of demand and the customers are added to the cheapest open facility with enough capacity. If no open facility has enough capacity, the

4.3.2.21 void SSCFLPsolver::Load (char ∗ FileName, int format)

Public function for loading data.

Public function for loading data describing an instance of the SSCFLP. Data is given as a data file.

Parameters

FileName	pointer to char array. Contains the path (relative or absolute) to the data file.
format	integer. Specifies the format of the data file. 0 = DiazFernandez. 1,2,3 = Holmberg, Yang and Gadegaard. Default is 1.

4.3.2.22 void SSCFLPsolver::Load (int nn, int mm, int ∗∗ cc, int ∗ ff, int ∗ dd, int ∗ ss)

Public function for loading data.

Public function for loading data describing an instance of the SSCFLP. Data is given directly.

Parameters

nn	integer. The number of facility sites.
mm	integer. The number of customers.
cc	pointer to a pointer to array of integers. cc[i][j] = cost of assigning customer j to facility i. Must have dimmensions at least n times m
ff	pointer to an array of integers. ff[i] = cost of opening facility at site i. Must have dimmensions at least n.
dd	pointer to an array of integers. dd[j] = demand at customer j. Must have dimmensions at least m.
ss	pointer to an array of integers. ss[i] = capacity at facility site i. Must have dimmensions at least n.

4.3.2.23 void SSCFLPsolver::Load (**rdDat** ∗ data)

Public function for loading the data of a SSCFLP.

Public function for loading data describing an instance of the SSCFLP. Data is given as a const reference to a rdDat object.

Parameters

data	pointer to a rdDat object containing the data of a SSCFLP

4.3.2.24 void SSCFLPsolver::PrintProgramInfo ( ) [private]

Prints info about the program and the author.

Prints information about the program and the author and the program. List of important parameter values is printed

4.3.2.25 void SSCFLPsolver::printStats ( )

Displays statistics for an instance of SSCFLP loaded into the SSCFLPsolver object.

Displays statistics for the instance of SSCFLP loaded into the SSCFLPsolver object. If the Load function has not been called, the method will just print rubbish

4.3.2.26 void SSCFLPsolver::RefineAllocation ( ) [private]

Performs local branching on the allocation varibales.

Method used in the RunLBHeur() function after the RefineLocation() function has been called.

Parameters

LBmod	reference to an IloModel object. Used to store a copy of the SSCFLP we are solving
LBcpx	reference to an IloCplex object. USed to solve the IloModel LBmod

4.3.2.27 void SSCFLPsolver::RefineLocation ( ) [private]

Performs local branching on the locational variables.

Method intended to be used by the RunLBHeur() function. It performs local branching on the locational variables. The first local branching constraints is based on the solution to the LP relaxation after adding cuts.

Parameters

LBmod	reference to an IloModel object. Used to store a copy of the SSCFLP we are solving
LBcpx	reference to an IloCplex object. USed to solve the IloModel LBmod

4.3.2.28 bool SSCFLPsolver::Run ( )

This is the main function in the SSCFLPsolver class.

This is the main function in the SSCFLPsolver class. The function executes the cutting plane algorithm as well as the cut and solve algorithm.

Returns

true if everything goes as planned. false otherwise.

4.3.2.29 bool SSCFLPsolver::RunAsHeuristic ( )

Runs a local branching heuristic.

Runs a local branching heuristic for the SSCFLP. First the a cutting plane algorithm is run. The y-variables which are positive in the cut-enhanced LP-relaxation is used to indicate which facilities are likely to be in an optimal solution. A local branching constraint is added and cplex is run for a limited amount of time. The solution found after this time-limited run is used as the first solution in a local branching constraint refining the locational deccision. That is, only local branching is performed on the locational variables. When a satisfactory solution is found, a second phase is entered where the allocation-decions is refined. The final solution can be accessed by getSolution(int∗y, int∗ x).

Returns

True if a feasible solution is found, and false otherwise.

4.3.2.30 void SSCFLPsolver::RunLBHeur ( ) [private]

Runs a simple local branching heuristic.

Runs a simple local branching heuristic that starts by refining the locational decision followed by a second phase where the allocation of customers is gradually improved

4.3.2.31 bool SSCFLPsolver::SepCuts ( ) [private]

Separates an LP solution (x,y) from the capacity constraints.

Separates an LP solution (x,y) from the capacity constraints. Tries first with a lifted cover inequality, then with an extended cover inequality and finally with a fenchel cut. CutsSeparated &&

CutsSeparated &&

4.3.2.32 void SSCFLPsolver::setC (int i, int j, int cc)

Sets the assignment cost c[i][j].

Sets the cost of assigning customer j to facility i

Parameters

i	integer. Index of the facility.
j	integer. Index of the customer.
cc	integer. Value of the assignment cost c[i][j] after execution.

Note

0<= i,j, i<getNumFac(), and j<getNumCust(). Note also that setC does not! change the objective of the IloCplex objects. It only changes the internal data.

4.3.2.33 void SSCFLPsolver::setDualAcentOn ( ) [inline]

Enables the dual ascent algorithm.

Enables the dual ascent algorithm for solving the sparse problems. It relaxes the assignment constraints in a semi-Lagrangean manner, and increases the dual multiplier until an optimal solution has been found.

4.3.2.34 void SSCFLPsolver::setF (int i, int ff) [inline]

Sets the fixed opening cost f[i].

Sets the cost of opening facility i

Parameters

i	integer. Index of the facility.
ff	integer. Value of the fixed cost f[i] after execution.

Note

0<= i < getNumFac(). Note also that setF does not! change the objective function coefficient of the IloCplex objects. It only changes the internal data.

4.3.2.35 bool SSCFLPsolver::setSolution (int nn, int mm, int * yval, int * xass, int UBval)

Hands a solution to the solver.

This function can be used to set a (heuristic) solution internally in cplex.

Parameters

nn	integer. Length of the array yval. Must be less than or equal to the number of facility sites.
mm	integer. Length of the array xass. Must be less than or equal to the number of customers.
yval	pointer to an array of integers. yval[i]=1 iff facility i is open in the solution you are providing.
xass	pointer to an array of integers. xass[j]=i iff customer j is assigned to facility i in the solution you provide.
UBval	integer. Contains the objective function value of the solution you provide. Will be used as cutoff value by cplex.

Returns

true if the solution provided was infact feasible.

Note

Should be called prior to calling run. Cplex is instructed to solve the problem as an LP with the solution given by (yval, xass) fixed.

4.3.2.36 bool SSCFLPsolver::solveSparseUsingDualAscent (double & NewObjective, **solution** & incumbent) `[private]`

A dual ascent algorithm for the sparse problems.

Algortihm that implements a semi-Lagrangean based dual ascent algorithm for solving the sparse problems. It start by writing the constraints $ {i I} x_{ij} =1$ as the two sets of constriants $ {i I} x_{ij} 1$ and $ {i I} x_{ij} 1$. The latter set is then first surrogate relaxed by multipliers $e=(1,,1)$ and then the resulting constriaint is relaxed in a Lagrangean manner by lagrangean multiplyer $$. This results in a Lagrangean dual having only one variable and a dual which closes the duality gap.

Parameters

NewObjective	reference to a double. Contains on output the new objective function value {if} it improves the current best solution value.

Returns

true if the dual ascent algorithm solved the problem to optimality. Note, we return false if we could not find an improving solution!

4.3.2.37 void SSCFLPsolver::twoSwap ( ) [private]

Two exchange local search. Swaps two customers if a gain is obtained

4.3.3 Member Data Documentation

4.3.3.1 IloRangeArray SSCFLPsolver::AssCst [private]

Holds the assignemnt constraints of the denseproblem only!

4.3.3.2 double SSCFLPsolver::avgC [private]

Contains the average value of the assignment costs after calling Load().

4.3.3.3 double SSCFLPsolver::avgD [private]

Contains the average value of the customer demands after calling Load().

4.3.3.4 double SSCFLPsolver::avgF [private]

Contains the average value of the fixed costs after calling Load().

4.3.3.5 double SSCFLPsolver::avgS [private]

Contains the average value of the facility capacities after calling Load().

4.3.3.6 **IloNumMatrix** SSCFLPsolver::c [private]

Assignment costs. c[i][j] = cost of assigning customer j to facility i.

4.3.3.7 double SSCFLPsolver::CnSTime [private]

CPU seconds used in the cut and solve phase.

4.3.3.8 bool SSCFLPsolver::CplexOutOff [private]

Used to flag that cplex' output should be redirected to the null stream.

4.3.3.9 double SSCFLPsolver::CutLowerBound [private]

The lower bound produced by the cutting plane algorithm.

4.3.3.10 double SSCFLPsolver::CutTime [private]

CPU seconds used in the cutting phase.

4.3.3.11 IloNumArray SSCFLPsolver::d [private]

Demands. d[j] = demand at customer j.

4.3.3.12 bool SSCFLPsolver::debugMe [private]

Used to flag if debug print outs should be enabled.

4.3.3.13 IloCplex SSCFLPsolver::DenseCplex [private]

The IloCplex environment used for the dense problem.

4.3.3.14 IloModel SSCFLPsolver::DenseModel [private]

The IloModel used for the dense problem.

4.3.3.15 IloObjective SSCFLPsolver::DenseObj [private]

Extractable holding the objective function of the dense problem.

4.3.3.16 bool SSCFLPsolver::displayStats [private]

Used to flag if statistics should be printed to screen.

4.3.3.17 IloNumArray SSCFLPsolver::duals [private]

Holds the duals of the assingment constraints of the dense problem.

4.3.3.18 IloEnv SSCFLPsolver::env [private]

The environment used throughout the lifetime of the object.

4.3.3.19 IloNumArray SSCFLPsolver::f [private]

Fixed opening costs. f[i] = cost of opening facility i.

4.3.3.20 std::vector<std::pair<int,int> > SSCFLPsolver::fixAfterCuttingPhase [private]

4.3.3.21 std::vector<std::pair<int,int> > SSCFLPsolver::fixBeforeCuttingPhase [private]

4.3.3.22 bool SSCFLPsolver::hasCleanedUp [private]

Used to flag if the CleanUp function has been called.

4.3.3.23 bool SSCFLPsolver::hasRun [private]

Used to flag if the Run() function has been called.

4.3.3.24 IloInt SSCFLPsolver::m [private]

Number of customers.

4.3.3.25 int SSCFLPsolver::maxC [private]

Contains the maximum value for the assignment costs after calling Load().

4.3.3.26 int SSCFLPsolver::maxD [private]

Contains the maximum value for the customer demands after calling Load().

4.3.3.27 int SSCFLPsolver::maxF [private]

Contains the maximum value for the fixed opening costs after calling Load().

4.3.3.28 int SSCFLPsolver::maxS [private]

Contains the maximum value for the facility capacities after calling Load().

4.3.3.29 int SSCFLPsolver::minC [private]

Contains the minimum value for the assignment costs after calling Load().

4.3.3.30 int SSCFLPsolver::minD [private]

Contains the minimum value for the customer demands after calling Load().

4.3.3.31 int SSCFLPsolver::minF [private]

Contains the minimum value for the fixed opening costs after calling Load().

4.3.3.32 int SSCFLPsolver::minS [private]

Contains the minimum value for the facility capacities after calling Load().

4.3.3.33 double SSCFLPsolver::myEpsilon [private]

My interpretation of "equal". Constructor initialized

4.3.3.34 double SSCFLPsolver::myOne [private]

My interpretation of "one". Constructor initialized.

4.3.3.35 double SSCFLPsolver::myZero [private]

My interpretation of "zero". Constructor initialized.

4.3.3.36 IloInt SSCFLPsolver::n [private]

Number of facility sites.

4.3.3.37 int* SSCFLPsolver::NumCuts [private]

Counts the number of cuts generated.

4.3.3.38 int SSCFLPsolver::ObjVal [private]

After the Run() has been called, ObjVal contains the optimal solution value.

4.3.3.39 double SSCFLPsolver::Runtime [private]

CPU seconds used for the whole procedure.

4.3.3.40 IloNumArray SSCFLPsolver::s [private]

Capacities; s[i] = capacity at customer i.

4.3.3.41 bool SSCFLPsolver::solveSparseUsingDualAscentAlg [private]

True if one should use dual ascent for solving the sparse problems. Defaults to false.

4.3.3.42 IloCplex SSCFLPsolver::SparseCplex [private]

The IloCplex environment used for the sparse problem.

4.3.3.43 IloModel SSCFLPsolver::SparseModel [private]

The IloModel used for the sparse problem.

4.3.3.44 IloObjective SSCFLPsolver::SparseObj [private]

Extractable holding the objective function of the sparse problem.

4.3.3.45 double SSCFLPsolver::StartRunTime [private]

Clock time where the Run() function is called.

4.3.3.46 double SSCFLPsolver::StartTime [private]

Clock time where the SSCFLPsolver object is initialized.

4.3.3.47 **testStats*** SSCFLPsolver::stats [private]

Pointer to an instance of the testStats struct. Contains on termination the statistics gathered during optimization.

4.3.3.48 IloInt SSCFLPsolver::TD [private]

Total demand. TD = sum_j d[j].

4.3.3.49 **IloVarMatrix** SSCFLPsolver::x

IloVarMatrix used for the assignment variables

4.3.3.50 int* SSCFLPsolver::xSol [private]

Pointer to an array of integers. If xSol[j]=i customer j is assigned to facility i.

4.3.3.51 IloNumVarArray SSCFLPsolver::y

IloNumVarArray used for the location variables.

4.3.3.52 int* SSCFLPsolver::ySol [private]

Pointer to an array of integers. If ySol[i]=1 facility i is open.

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

- SSCFLPsolver.h
- SSCFLPsolver.cpp

4.4 testStats Struct Reference

```
#include <SSCFLPsolver.h>
```

Public Attributes

- int n

  Number of facilities.
- int m

  Number of customers.
- long numberOfIterations

  Number of iterations in the dual ascent algorithm.
- double time

  TotalTime.
- long itWhereOptWasFound

  The dual ascent iteration where the optimal solution was found. If equal to zero, optimal solution was found by initial heuristic.
- double initialUpperBound

  The initial upper bound before dual ascent starts.
- double bestUpperBound

  Best upper bound. Equal to optimal solution if no optimality gap is left.
- double bestLowerBound

  Best lower bound on the instance.
- double percentageGap

  *Percentage gap between lower and upper bound. Calculated as (UB - LB) / LB * 100.*
- double CuttingTime

  Time used in the cutting plane algorithm.
- double CutAndSolveTime

  Time used in the cut and solve algorithm.
- double avgNumOfDualItPerSparse

  The average number of dual ascent iterations per sparse problem.
- double numOfSparseSolved

  Total number of sparse problems solved.
- double WeakLowerBound

Lower bound before adding cutting planes.

- double LowerBoundAfterCusts

  Lower bound after applying the cutting plane algorithm.
- double avgPercentLeft

  Average number of assingment variables left after solving Sparse using dual ascent.

4.4.1 Member Data Documentation

4.4.1.1 double testStats::avgNumOfDualItPerSparse

The average number of dual ascent iterations per sparse problem.

4.4.1.2 double testStats::avgPercentLeft

Average number of assingment variables left after solving Sparse using dual ascent.

4.4.1.3 double testStats::bestLowerBound

Best lower bound on the instance.

4.4.1.4 double testStats::bestUpperBound

Best upper bound. Equal to optimal solution if no optimality gap is left.

4.4.1.5 double testStats::CutAndSolveTime

Time used in the cut and solve algorithm.

4.4.1.6 double testStats::CuttingTime

Time used in the cutting plane algorithm.

4.4.1.7 double testStats::initialUpperBound

The initial upper bound before dual ascent starts.

4.4.1.8 long testStats::itWhereOptWasFound

The dual ascent iteration where the optimal solution was found. If equal to zero, optimal solution was found by initial heuristic.

4.4.1.9 double testStats::LowerBoundAfterCusts

Lower bound after applying the cutting plane algorithm.

4.4.1.10 int testStats::m

Number of customers.

4.4.1.11 int testStats::n

Number of facilities.

4.4.1.12 long testStats::numberOfIterations

Number of iterations in the dual ascent algorithm.

4.4.1.13 double testStats::numOfSparseSolved

Total number of sparse problems solved.

4.4.1.14 double testStats::percentageGap

Percentage gap between lower and upper bound. Calculated as (UB - LB) / LB $*$ 100.

4.4.1.15 double testStats::time

TotalTime.

4.4.1.16 double testStats::WeakLowerBound

Lower bound before adding cutting planes.

The documentation for this struct was generated from the following file:

- SSCFLPsolver.h

4.5 TFENCHELopt Struct Reference

```
#include <vico.h>
```

Public Attributes

- int maxit
- int sg_strat
- double alpha
- int H
- int CHK
- int Algo
- int Freq
- int Reduce

4.5.1 Member Data Documentation

4.5.1.1 int TFENCHELopt::Algo

4.5.1.2 double TFENCHELopt::alpha

4.5.1.3 int TFENCHELopt::CHK

4.5.1.4 int TFENCHELopt::Freq

4.5.1.5 int TFENCHELopt::H

4.5.1.6 int TFENCHELopt::maxit

4.5.1.7 int TFENCHELopt::Reduce

4.5.1.8 int TFENCHELopt::sg_strat

The documentation for this struct was generated from the following file:

- vico.h

4.6 VICcut Struct Reference

Implements seperations routines several different problems.

`#include <vico.h>`

Public Attributes

- char sense

    sense of inequality: 'L' means <=, 'G' is >=
- double rhs

    right-hand side of inequality
- int nzcnt

    number of non-zeros in the inequality
- int * nzind

    column/variable indices of the non-zeros
- double * nzval

    values of the non-zeros in the inequality
- double UsrRVal

    may be used to store e.g. a dual variable
- int UsrIVal

    may be used to store e.g. an integer flag
- void * UsrDatPtr

    pointer to any additional user data
- struct VICcut * nextcut

    pointer to next cut

4.6.1 Detailed Description

Implements seperations routines several different problems.

This is the header file for module "vico.c" (Valid Inequalities for selected Combinatorial Optimization problems)

Author

Andreas Klose

Version

2.9.0

Note

Programming language: C
In order to use the functions listed below from within a SUN Pascal program compiled with option -L using the SUN Pascal compiler, compile file vico.c with option -DSUNPAS and link with libF77

4.6.2 License

If you use the software in any academic work, please make a reference to "An LP-Based Heuristic for Two--Stage Capacitated Facility Location Problems", Journal of the Operational Research Society, Vol. 50, No. 2 (Feb. 1999), pp. 157-166.

4.7 Change log for vico.h

FILE:	vico.h and vico.c		
Version:	2.9.0		
- -			
CHANGE LOG:	DATE	VER.-NO.	CHANGES MADE
	2003–05–12	1.0.0	first implementation
	2003-07-07	1.1.0	bug in VICuflohi removed
	2003-08-06	2.0.0	routine VIClci and VICkconf added
	2003-08-07	2.1.0	routine VICgapfn added
	2003-08-15	2.2.0	routine VICuflcov added
	2003-08-22	2.3.0	VIClci, VICkconf modified to ease handling
	2003-08-23	2.3.1	bugs in VICuflohi removed (shortest path computation now based on FIFO, error in cut existence check removed)
	2003-08-23	2.3.2	minor changes, some "statics" inserted
	2003-08-27	2.3.3	minor changes: routines VICsearchcut, VICclearlst added
	2004-02-20	2.4.0	routine VICeci added
	2004-02-24	2.4.1	routine VICintsort, VICdblsort added
	2004-02-25	2.5.0	routine VICkpreduce added
	2004-03-11	2.5.1	small bug in VIClci removed
	2005-05-10	2.5.2	bug in VICkconf "(if card==n) ... return" removed
	2005-08-01	2.5.3	bug in kirestore() removed (all coefficients of a generated cut were modified if at least one variable was inverted!)
	2007-05-30	2.6.0	Started with implementing Fenchel cuts based on single-node flow structures
	2007-06-05	2.6.1	Different subgradient strategies for Fenchel cut generation implemented.
	2007-06-14	2.6.2	Different algorithms for solving the subproblem within Fenchel cut generation may optionally be chosen
	2007-06-15	2.6.3	Additional parameter "Freq" for Fenchel cut generation introduced
	2007-06-22	2.7.0	Additional parameter "Reduce" for Fenchel cut generation introduced. If Reduce=0 all "zero arcs" are removed and the inequality is generated for the reduced polytope
	2007-08-20	2.7.1	small bug in vicsnffenchel removed: capacities are now allowed to be zero (variable can then be ignored)
	2008-01-09	2.8.0	start to implement routine for exact knapsack separation
	2008-06-18	2.8.1	Some bugs removed in VICkplift and VICkpsep
	2008-06-19	2.8.2	Bug remove in VICkplift ((t-pi) could be negative)
	2012-08-20	2.8.3	Adjusted the uplifting in VICkplift and included possibility to exclusively fix variables of zero LP value when defining the reduced knapsack polytope in the exact knapsack separation procedure
	2012-12-18	2.9.0	Inclusion of a procedure suggested by Kaparis and Letchford (2010) to separate extended cover inequalities

4.7.1 Member Data Documentation

4.7.1.1 struct **VICcut**∗ VICcut::nextcut

pointer to next cut

4.7.1.2 int VICcut::nzcnt

number of non-zeros in the inequality

4.7.1.3 int* VICcut::nzind

column/variable indices of the non-zeros

4.7.1.4 double* VICcut::nzval

values of the non-zeros in the inequality

4.7.1.5 double VICcut::rhs

right-hand side of inequality

4.7.1.6 char VICcut::sense

sense of inequality: 'L' means <=, 'G' is >=

4.7.1.7 void* VICcut::UsrDatPtr

pointer to any additional user data

4.7.1.8 int VICcut::UsrIVal

may be used to store e.g. an integer flag

4.7.1.9 double VICcut::UsrRVal

may be used to store e.g. a dual variable

The documentation for this struct was generated from the following file:

- vico.h

Chapter 5

File Documentation

5.1 main.cpp File Reference

```
#include "SSCFLPsolver.h"
#include <chrono>
```

Functions

- int main (int argc, char **argv)

5.1.1 Function Documentation

5.1.1.1 int main (int argc, char ** argv)

5.2 rdDat.cpp File Reference

```
#include "rdDat.h"
```

5.3 rdDat.h File Reference

```
#include <random>
#include <exception>
#include <stdexcept>
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <string>
```

Classes

- class rdDat

5.4 solution.cpp File Reference

```
#include "solution.h"
```

5.5 solution.h File Reference

```
#include <exception>
#include <stdexcept>
#include <iostream>
#include <algorithm>
```

Classes

- class solution

5.5.1 Detailed Description

Author

Sune Lauth Gadegaard

Date

2015–04-09

Version

1.0.0

Class for storing a solution to the single source capacitated facility location problem. Implemented in C++.

5.6 SSCFLPsolver.cpp File Reference

```
#include "SSCFLPsolver.h"
```

Functions

- double roundToTwo (double d)

  Rounds double to two digits.
- double calcPercent (double ub, double lb)

  Calculates the percentage gab between ub and lb.
- ILOUSERCUTCALLBACK1 (KnapsackSep, SSCFLPsolver &, solver)

Variables

- bool initializeWithDual1 = false
- bool initializeWithPercent1 = true
- bool initializeWithSolution1 = false
- long cutsAddedInCutCallback = 0
- long doubleKPcutsAdded = 0

5.6.1 Function Documentation

5.6.1.1 double calcPercent (double ub, double lb)

Calculates the percentage gab between ub and lb.

Parameters

ub	Double. A number larger than lb, for example an upper bound on SSCFLP.
lb	Double. A number smaller than ub, for example a lower bound on SSCFLP.

5.6.1.2 ILOUSERCUTCALLBACK1 (KnapsackSep , **SSCFLPsolver** & , solver)

5.6.1.3 double roundToTwo (double d)

Rounds double to two digits.

Parameters

d	Double. The double which should be rounded.

5.6.2 Variable Documentation

5.6.2.1 long cutsAddedInCutCallback = 0

5.6.2.2 long doubleKPcutsAdded = 0

5.6.2.3 bool initializeWithDual1 = false

5.6.2.4 bool initializeWithPercent1 = true

5.6.2.5 bool initializeWithSolution1 = false

5.7 SSCFLPsolver.h File Reference

```
#include <ilcplex/ilocplex.h>
#include <exception>
#include <stdexcept>
#include <vector>
#include "vico.h"
#include "combo.h"
#include "solution.h"
#include <time.h>
#include <chrono>
#include "rdDat.h"
```

Classes

- struct testStats
- class SSCFLPsolver

Typedefs

- typedef
  std::chrono::high_resolution_clock CPUclock
- typedef IloArray< IloNumVarArray > IloVarMatrix

    An IloArray of IloNumVarArrays.
- typedef IloArray< IloNumArray > IloNumMatrix

    An IloArray of IloNums.

5.7.1 Typedef Documentation

5.7.1.1 typedef std::chrono::high_resolution_clock **CPUclock**

5.7.1.2 typedef IloArray<IloNumArray> **IloNumMatrix**

An IloArray of IloNums.

5.7.1.3 typedef IloArray<IloNumVarArray> **IloVarMatrix**

An IloArray of IloNumVarArrays.

5.8 vico.h File Reference

`#include <ilcplex/cplex.h>`

Classes

- struct VICcut

    Implements seperations routines several different problems.
- struct TFENCHELopt

Macros

- #define SG_DEFAULT 0
- #define SG_DEFLECT 1
- #define SG_SMOOTH 2
- #define ALG_MT1 0
- #define ALG_DP 1
- #define ALG_CPLEX 2

Typedefs

- typedef struct VICcut VICcut

    Implements seperations routines several different problems.
- typedef struct TFENCHELopt TFENCHELopt

    The following is used for defining parameters for Fenchel cut generation.

Functions

- int VICisintvec (int n, double ∗pi)
- void VICsetdefaults ()
- void VICaddtolst (VICcut ∗∗first, VICcut ∗firstnew)
- void VICfreecut (VICcut ∗∗cut)
- void VICfreelst (VICcut ∗∗first)
- char VICallocut (int numnz, VICcut ∗∗cut)
- VICcut ∗ VICsearchcut (VICcut ∗cutlst, VICcut ∗cut)
- void VICclearlst (VICcut ∗∗first)
- void VICsort (int n, int ascending, int doinit, int size, void ∗numbers, int ∗order)
- void VICcflfc (int m, int n, int ∗demand, int ∗capaci, double ∗x, double ∗y, VICcut ∗∗fc_cut)
- void VICcflsmi (int m, int n, int ∗demand, int ∗capaci, double ∗x, double ∗y, VICcut ∗∗first_smi)
- void VICuflohi (int m, int n, double ∗x, double ∗y, VICcut ∗∗first_ohi)
- void VICuflcov (CPXENVptr Env, int m, int n, char SolveCov, double ∗x, double ∗y, VICcut ∗∗first_cut)
- void VICkpreduce (int WHATRED, int TRYCLI, int n, int ∗cap, char sense, int ∗weight, int ∗order, int ∗indx, VICcut ∗∗clique)
- void VIClci (int n, int cap, char sense, int ∗weight, int ∗indx, double ∗xlp, double ∗rco, VICcut ∗∗lci)
- void VICkconf (int n, int cap, char sense, int ∗weight, int ∗indx, double ∗xlp, double ∗rco, VICcut ∗∗kconf)
- void VICeci (int n, int cap, char sense, int ∗weight, int ∗order, int ∗indx, double ∗xlp, VICcut ∗∗eci)
- void VICecikl (int n, int cap, int do_exact, char sense, int ∗weight, int ∗indx, double ∗xlp, VICcut ∗∗eci)
- void VICgapfn (int m, int n, int IsGap, int ∗capaci, int ∗∗weight, double ∗X, int ∗indx, VICcut ∗∗fn_cut)
- void VICkpsep (CPXENVptr Env, int justUp, int n, int cap, char sense, int ∗weight, int ∗indx, double ∗xlp, double ∗rco, VICcut ∗∗cut)

5.8.1 Macro Definition Documentation

5.8.1.1 #define ALG_CPLEX 2

5.8.1.2 #define ALG_DP 1

5.8.1.3 #define ALG_MT1 0

5.8.1.4 #define SG_DEFAULT 0

5.8.1.5 #define SG_DEFLECT 1

5.8.1.6 #define SG_SMOOTH 2

5.8.2 Typedef Documentation

5.8.2.1 typedef struct **TFENCHELopt TFENCHELopt**

The following is used for defining parameters for Fenchel cut generation.

5.8.2.2 typedef struct **VICcut VICcut**

Implements seperations routines several different problems.

This is the header file for module "vico.c" (Valid Inequalities for selected Combinatorial Optimization problems)

Author

Andreas Klose

Version

2.9.0

Note

Programming language: C
In order to use the functions listed below from within a SUN Pascal program compiled with option -L using the SUN Pascal compiler, compile file vico.c with option -DSUNPAS and link with libF77

5.8.3 License

If you use the software in any academic work, please make a reference to "An LP-Based Heuristic for Two-- Stage Capacitated Facility Location Problems", Journal of the Operational Research Society, Vol. 50, No. 2 (Feb. 1999), pp. 157-166.

5.9 Change log for vico.h

FILE:	vico.h and vico.c		
Version:	2.9.0		
- -			
CHANGE LOG:	DATE	VER.-NO.	CHANGES MADE
	2003–05–12	1.0.0	first implementation
	2003-07-07	1.1.0	bug in VICuflohi removed
	2003-08-06	2.0.0	routine VIClci and VICkconf added
	2003-08-07	2.1.0	routine VICgapfn added
	2003-08-15	2.2.0	routine VICuflcov added
	2003-08-22	2.3.0	VIClci, VICkconf modified to ease handling
	2003-08-23	2.3.1	bugs in VICuflohi removed (shortest path computation now based on FIFO, error in cut existence check removed)
	2003-08-23	2.3.2	minor changes, some "statics" inserted
	2003-08-27	2.3.3	minor changes: routines VICsearchcut, VICclearlst added
	2004-02-20	2.4.0	routine VICeci added
	2004-02-24	2.4.1	routine VICintsort, VICdblsort added
	2004-02-25	2.5.0	routine VICkpreduce added
	2004-03-11	2.5.1	small bug in VIClci removed
	2005-05-10	2.5.2	bug in VICkconf "(if card==n) ... return" removed
	2005-08-01	2.5.3	bug in kirestore() removed (all coefficients of a generated cut were modified if at least one variable was inverted!)
	2007-05-30	2.6.0	Started with implementing Fenchel cuts based on single-node flow structures
	2007-06-05	2.6.1	Different subgradient strategies for Fenchel cut generation implemented.
	2007-06-14	2.6.2	Different algorithms for solving the subproblem within Fenchel cut generation may optionally be chosen
	2007-06-15	2.6.3	Additional parameter "Freq" for Fenchel cut generation introduced
	2007-06-22	2.7.0	Additional parameter "Reduce" for Fenchel cut generation introduced. If Reduce=0 all "zero arcs" are removed and the inequality is generated for the reduced polytope
	2007-08-20	2.7.1	small bug in vicsnffenchel removed: capacities are now allowed to be zero (variable can then be ignored)
	2008-01-09	2.8.0	start to implement routine for exact knapsack separation
	2008-06-18	2.8.1	Some bugs removed in VICkplift and VICkpsep
	2008-06-19	2.8.2	Bug remove in VICkplift ((t-pi) could be negative)
	2012-08-20	2.8.3	Adjusted the uplifting in VICkplift and included possibility to exclusively fix variables of zero LP value when defining the reduced knapsack polytope in the exact knapsack separation procedure
	2012-12-18	2.9.0	Inclusion of a procedure suggested by Kaparis and Letchford (2010) to separate extended cover inequalities

5.9.1 Function Documentation

5.9.1.1 void VICaddtolst (**VICcut** ∗∗ first, **VICcut** ∗ firstnew)

PURPOSE: Adds a linked list of cuts to an existing linked list of cuts. The new cuts are inserted at the top of the existing list of cuts. Let

First -> Second -> ... -> Last -> NULL

be the list of already existing cuts and let

Firstnew -> Secondnew -> ... -> Lastnew -> NULL denote the linked list of new cuts. After calling the procedure, the old list looks like

Firstnew -> Secondnew -> ... > Lastnew -> First -> Second -> Last -> NULL

However, the memory allocated for the cuts in the new list is not copied! Therefore, do not delete it after addition to the old list.

Parameters

first	: pointer to the pointer to first cut in the existing list of cuts (if the list is empty *first must be NULL)
firstnew	: pointer to first cut in the new list of cuts (which may consists of just one cut)

```
VICcut *MyCutsSoFar = NULL;
VICcut *MyNewCuts = NULL;

ProcedureForGeneratingNewCuts( &MyNewCuts );
if ( MyNewCuts != NULL ) VICaddtolst( &MyCutsSoFar, MyNewCuts );
```

5.9.1.2 char VICallocut (int numnz, **VICcut** ** cut)

Allocates memory required to store data of a cut

Parameters

numnz	number of nonzero coefficients in the cut
cut	on completion *cut is the pointer to the cut

Returns

1 on success and 0 otherwise

```
VICcut* MyCutPtr;
int     numnz=1000;
VICallocut( numnz, &MyCutPtr );
```

5.9.1.3 void VICcflfc (int m, int n, int * demand, int * capaci, double * x, double * y, **VICcut** ** fc_cut)

Tries to generate a (single) extended flow cover inequality for the CFLP, which is violated by the current solution (x,y). The procedure may also be used to generate extended flow cover inequalities for the single-node flow problem given by

$$\sum_j z_j = d \tag{5.1}$$
$$z_j \leq capaci_j * y_j \tag{5.2}$$
$$0 \leq z_j \leq \min\{d, capaci_j\} \tag{5.3}$$
$$y_j \in \{0,1\}, \quad \forall j \tag{5.4}$$
$$\tag{5.5}$$

In this case, call the procedure with m=1, demand = d, x=z/d

Parameters

n	number of potential depot sites
m	number of customers

demand	pointer to an array of integers of size of at least m containing the customers' demands
capaci	pointer to an array of integers of size of at least n containing the depot capacities
x	pointer to an array of doubles of size of at leat m∗n containing the allocation part of the fractional solution, which should be separated by a flow cover inequality. Let i=0,...,m-1 and j=0,...n-1 be the indices of customers and depot sites, resp. Then x[i∗n + j] is the solution value of the allocation variable x(i,j), where 0 <= x(i,j) <= 1. The variable x(i,j) denotes the fraction of customer i's demand met from facility j.
y	pointer to an array of doubles of size of at least n containing the location part of the fractional solution, which should be separated by a flow cover inequality. 0 <= y[j] <= 1 and x(i,j) <= y[j]
fc_cut	pointer to a pointer to a cut. If no violated flow cover is found, the null pointer is returned; otherwise ∗∗fc_cut contains the cut.fc_cut->nzval contains the nonzeros of the cut, and fc_cut->nzind contains the column indices of the nonzeros, where the index of value i∗n+j corresponds to the allocation variable x(i,j) and the index m∗n+j corresponds to the location variable y(j).

```
VICcut* MyCutList = NULL;
VICcut* MyNewCut = NULL;
int m, n, *demand=NULL, *capaci=NULL;

Read_Problem_Data_and_Allocate_Space( m, n, demand, capaci, ... );
Solve_Something_like_the_LP_Relaxation_and_obtain_x_y;

VICcflfc( m, n, demand, capaci, x, y, &MyNewCut );
if ( MyNewCut != NULL ) VICaddtolst( &MyCutList, MyNewCut );
```

5.9.1.4 void VICcflsmi (int m, int n, int ∗ demand, int ∗ capaci, double ∗ x, double ∗ y, **VICcut** ∗∗ first_smi)

Generates special types of "submodular inequalities" for the CFLP using a separation heuristic of Aardal. Several such inequalities, which cut off the solution (x,y), may be returned.

Parameters

n	number of potential depot sites
m	number of customers
demand	pointer to an array of integers of size of at least m containing the customers' demands
capaci	pointer to an array of integers of size of at least n containing the depot capacities
x	pointer to an array of doubles of size of at leat m∗n containing the allocation part of the fractional solution, which should be separated by a submodular inequality. Let i=0,...,m-1 and j=0,...n-1 be the indices of customers and depot sites, resp. Then x[i∗n + j] is the solution value of the allocation variable x(i,j), where 0 <= x(i,j) <= 1. The variable x(i,j) denotes the fraction of customer i's demand met from facility j.

```
VICcut* MyCutList = NULL;
VICcut* MyNewCut = NULL;
int m, n, *demand=NULL, *capaci=NULL;

Read_Problem_Data_and_Allocate_Space( m, n, demand, capaci, ... );
Solve_Something_like_the_LP_Relaxation_and_obtain_x_y;
VICcflsmi( m, n, demand, capaci, x, y, &MyNewCut );
if ( MyNewCut != NULL ) VICaddtolst( &MyCutList, MyNewCut );
```

5.9.1.5 void VICclearlst (**VICcut** ∗∗ first)

Eliminates duplicate cuts from a linked list of cuts such that every cut is only contained once in that list

Parameters

first	pointer to the pointer to the first cut in the list to be cleared

5.9.1.6 void VICeci (int n, int cap, char sense, int ∗ weight, int ∗ order, int ∗ indx, double ∗ xlp, **VICcut** ∗∗ eci)

Separation procedure of Gabrel & Minoux for finding most violated extended cover inequality. See: V. Gabrel, M. Minoux (2002). A scheme for exact separation of extended cover inequalities and application to multidimensional knapsack problems. Oper. Res. Lett. 30:252-264. For an example see VIClci

Parameters

n	number of (free) variables appearing in the knapsack inequality.
cap	right-hand side of the knapsack inequality.
sense	sense (that is 'L' for <= or 'G' for >=) of the knapsack inequality.
weight	coefficient a_j of (free) variables in the knapsack inequality (coefficients a_j are not restricted to be nonnegative!).
order	NULL or an ordering of the items in the knapsack according to increasing weights.
indx	indices of the (free) variables in the knapsack inequality. If null it is assumed that variables are numbered from 0 to n-1.
xlp	LP solution of (free) variables appearing in the knapsack inequality.
eci	pointer to the generated cut pointer.

5.9.1.7 void VICecikl (int n, int cap, int do_exact, char sense, int ∗ weight, int ∗ indx, double ∗ xlp, **VICcut** ∗∗ eci)

Procedure for generating extended cover inequalities as suggested in K. Kaparis, A.N. Letchford (2010). Separation algorithms for 0-1 knapsack polytopes, Math. Prog. 124:69-91.

Parameters

n	number of variables appearing in the knapsack inequality.
cap	right-hand side of the knapsack inequality.
do_exact	if equal to 1, exact separation is tried. This requires to repeatedly solve a 0-1 knapsack problem. If equal to 0 these knapsack problems are solved heuristically using a greedy method.
sense	sense (that is 'L' for <= or 'G' for >=) of the knapsack inequality
weight	coefficient a_j of variables in the knapsack inequality
indx	indices of the variables in the knapsack inequality. If NULL, it is assumed that variables are numbered from 0 to n-1.
xlp	LP solution of (free) variables appearing in the knapsack inequality
eci	pointer to the generated cut pointer (NULL if none found)

5.9.1.8 void VICfreecut (**VICcut** ∗∗ cut)

Frees the memory allocated for a cut to which the pointer ∗cut points

Parameters

cut	pointer to the pointer to the

5.9.1.9 void VICfreelst (**VICcut** ∗∗ first)

Frees the memory allocated for a linked list of cuts and empties the list

Parameters

first	: pointer to the pointer to the first cut in the list

```
VICcut* MyCutList;
many very strong and helpful cuts generated and optimum proven
VICfreelst( &MyCutList );
```

5.9.1.10 void VICgapfn (int m, int n, int IsGap, int * capaci, int ** weight, double * X, int * indx, **VICcut** ** fn_cut)

Given the GAP (or alternatively LEGAP)

$$\max \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{5.6}$$

$$s.t.: \sum_{i \in I} x_{ij} = 1, \quad \forall j \in J \tag{5.7}$$

$$\sum_{j \in J} w_{ij} x_{ij} \leq s_i, \quad \forall i \in I \tag{5.8}$$

$$x_{ij} \in \{0,1\} \forall i \in I, j \in J \tag{5.9}$$

the procedure uses a heuristic described in Farias IR, Nemhauser GL (2001), a family of inequalities for the generalized assignment polytope, Oper. Res. Letters 29, for finding a violated inequality of the form

$$\sum_{j \in J_i} w_{ij} x_{ij} + \sum_{j \in J_i} a_j \sum_{k \neq i} x_{kj} \leq s_i \tag{5.10}$$

where $J_i \subseteq J, a_j = s_i - (W_i - w_{ij})$ and $W_i = \sum_{j \in J_i} w_{ij}$

Parameters

m	number of agents.
n	number of jobs.
IsGap	set equal to 1 if the problem is a GAP, that is if every job has to be assigned to exactly one agent. Otherwise a LEGAP is assumed, that is some jobs may be not assigned to any agent.
capaci	pointer to an array of integers of length of at least m containing the agents' capacities.
weight	pointer to an array of pointers of length of at least m such that weight[i][j] gives the amount of resources required by agent i to perform job j.
X	pointer to an array of doubles of length of at least m∗n containing the fractional LP solution, X[i∗n+j] must contain the LP value of assignment variable x_{ij}
indx	if not NULL this array gives the indices of the assignment variables in the user's problem formulation, that is indx[i∗n+j] is the index of variable x_{ij}. If indx==NULL it is assumed that variables are numbered from 0 to m∗n-1, where i∗n+j is the index of variable x_{ij}.
fn_cut	pointer to the first cut in a linked list of generated cuts of this type

5.9.1.11 int VICisintvec (int n, double * pi)

Checks if a vector is almost integer

Parameters

n	integer. Size of the array
pi	pointer to double array. Contains the values of the array

5.9.1.12 void VICkconf (int n, int cap, char sense, int * weight, int * indx, double * xlp, double * rco, **VICcut** ** kconf)

Tries to get a (1,k)-configuration inequality using the separation heuristic of Crowder, Johnson, Padberg in Oper. Res. 31 (1983). For an alternative separation heuristic for (1,k)-configurations see Carlos E. Ferreira (1997). On Combinatorial Optimization Problems arising in Computer Systems Design. Phd Thesis, Technische Universit Berlin. Given the Knapsack-Polytop $X = x : \sum_{j \in N} w[j] * x[j] <= c, x_j = 0, 1$ a (1,k)-configuration is a set $NP \cup z$, where $NP \subset N$, such that

1. $\sum_{j \in NP} w[j] <= c$
2. The set $K \cup z$ is a cover with respect to N for all subsets K of NP with cardinality k

The corresponding (1,k)-configuration inequality is given by

$$(r-k+1)x[z] + \sum_{j \in NP} x[j] <= r, wherer = |NP| \tag{5.11}$$

Crowder, Johnson, Padberg propose the following separation heuristic:

1. 1. Let $S \subset N$ be the cover, and $z \in S$ the item with maximum weight. Set $NP = S - z$ and $k = |NP|$.
2. For all $j \in N - S$ with $\sum_{l \in N-S} w[l] <= c$ do:
   (a) Check if $K \cup z$ is a cover for any $K \subseteq NP$ with $|K| = k$
   (b) If this is the case build the corresponding (1,k)-configuration inequality and lift it.
   (c) If the lifted inequality is violated, add the inequality

Parameters

n	number of (free) variables appearing in the knapsack inequality.
cap	right-hand side of the knapsack inequality.
sense	sense (that is 'L' for <= or 'G' for >=) of the knapsack inequality
weight	coefficient a_j of (free) variables in the knapsack inequality (coefficients a_j are not restricted to be nonnegative!).
indx	indices of the (free) variables in the knapsack inequality. If null it is assumed that variables are numbered from 0 to n-1.
xlp	LP solution of (free) variables appearing in the knapsack inequality
rco	absolute values of reduced costs of variables appearing in the knapsack inequality
kconf	pointer to the first cut in a linked list of generated k-configuration cuts

5.9.1.13 void VICkpreduce (int WHATRED, int TRYCLI, int n, int * cap, char sense, int * weight, int * order, int * indx, **VICcut** ** clique)

Clique generation and coefficient improvement (reduction) for a single knapsack inequality of the form $\sum_j a_j x_j <= bor \sum_j a_j x_j >= b$ where $x_j = 0, 1$ for all j The procedure tries to generate a single clique inequality from the knapsack inequality and to reduce the right-hand side together with the coefficients of variables in the clique.

Parameters

WHATRED	Determines which coeffiecient improvement scheme is applied. WHATRED = 0 -> no coefficient improvement at all. WHATRED = 1 -> just simple coefficient improvement (increase w_j to c if x_j=1 implies all other x_k = 0). WHATRED = 2 -> only apply reduction of coefficients of variables in the clique. WHAT_RED = 3 -> do both types of improvement if possible.
TRYCLI	If TRYCLI=0 no cliques are generated, otherwise cliques are derived if possible. (In order to use the coefficient reduction, that is WHATRED >= 2; a clique is required and TRYCLI must equal 1).
n	number of (free) variables in the knapsack.
cap	pointer to an integer containing the capacity of the knapsack (right-hand side of inequality). ∗cap is possible changed (reduced).
sense	sense of inequality ('L' for <= and 'G' for >=).
weight	pointer to an array of integers of length of at least n containig the coefficients ("weights") of (free) variables in the inequality. Some of the weights[j] may be changed (reduced).
order	NULL or a pointer to an array of integers of length of at least n containing an ordering of the items 0,...,n-1 according to increasing weights.
indx	NULL or a pointer to an array of integers of length of at least n containing indices of the variables in the inequality. If indx=NULL, it is assumed that the variables are numbered from 0, ..., n-1
clique	pointer to the the generated clique cut

5.9.1.14 void VICkpsep (CPXENVptr Env, int justUp, int n, int cap, char sense, int ∗ weight, int ∗ indx, double ∗ xlp, double ∗ rco, **VICcut** ∗∗ cut)

Subroutine for exact knapsack separation. Given a fractional solution x∗, the most violated valid inequality $\pi x \leq 1$ is returned by optimizing over the 1-polar of the knapsack polytope, that is by solving the separation problem

$$\max\{\pi x^* : \pi x^t \leq 1 foreach feasible solution x^t\}. \tag{5.12}$$

The dual of the this separation problem is solved by means of column generation. In order to reduce the effort, the separation is performed for a reduced knapsack polytope obtained by resp. fixing variables x_j to 0 and 1 if $x_j^* = 0$ or $x_j^* = 1$. Afterwards, a sequential lifting of fixed variables is applied in order to get a valid cut.

Parameters

Env	pointer to the CPLEX environment as returned by CPXopenCPLEX. Env must be a valid pointer to the open CPLEX environment
justUp	if equal to one, only variables showing a value of zero in the LP solution are first fixed to zero. The resulting inequality is then just uplifted to obtain a (strengthened) inequality for the full polytope. Otherwise (justUp=0) both variables showing LP-value of zero and one are fixed. The resulting inequality is then first to be downlifted to obtain a valid inequality, and thereafter uplifting is applied.
n	number of variables appearing in the knapsack inequality.
cap	right-hand side of the knapsack inequality.
sense	sense (that is 'L' for <= or 'G' for >=) of the knapsack inequality.
weight	coefficient a_j of variables in the knapsack inequality (coefficients a_j are not restricted to be nonnegative!).
indx	indices of the variables in the knapsack inequality. If NULL it is assumed that variables are numbered from 0 to n-1
xlp	LP solution of variables appearing in the knapsack inequality.
rco	absolute values of reduced costs of variables appearing in the knapsack inequality.

cut	pointer to the generated cut pointer.

5.9.1.15 void VIClci (int n, int cap, char sense, int ∗ weight, int ∗ indx, double ∗ xlp, double ∗ rco, **VICcut** ∗∗ lci)

Heuristic procecedure of Gu, Nemhauser and Savelsbergh for generating lifted cover inequalties from a knapsack structure like

$$\sum_{j \in N} a_j x_j \leq b \tag{5.13}$$

$$0 \leq x_j \leq 1 \forall j \in N \tag{5.14}$$

$$x_j \in \{0,1\} \forall j \in N \tag{5.15}$$

or

$$\sum_{j \in N} a_j x_j \geq b \tag{5.16}$$

$$0 \leq x_j \leq 1 \forall j \in N \tag{5.17}$$

$$x_j \in \{0,1\} \forall j \in N \tag{5.18}$$

See: Gu Z, Nemhauser GL, Savelsbergh MWP (1998). "Lifted cover inequalities for 0-1 linear programs: Computation". INFORMS J. on Computing 10:427-437 EXAMPLE: Consider the following MIP

$$\max 7x_0 + 3x_1 + 6x_2 + 9x_3 + 10x_4 + 6x_5 + 8x_6 + 9x_7 \tag{5.19}$$

$$s.t. : x_0 + 2x_1 + x_2 \leq 2 \tag{5.20}$$

$$3x_3 + 5x_4 + 2x_5 + 6x_6 + 7x_7 \leq 11 \tag{5.21}$$

$$x_j \in \{0,1\}, \forall j \tag{5.22}$$

Furthermore, assume that variables x(3) is fixed to one and variable x_5 is fixed to zero. In order to derive a lifted cover inequality from the second inequality, the procedure above has to be called in the following way:

n = 3, cap = 11-3 = 8, sense = 'L', weight = (5, 6, 7), indx = (4, 6, 7), xlp = (x_4, x_6, x_7), rco = (|redcost(4)|, |redcost(6)|, |redcost(7)|)

Parameters

n	number of (free) variables appearing in the knapsack inequality.
cap	right-hand side of the knapsack inequality.
sense	sense (that is 'L' for <= or 'G' for >=) of the knapsack inequality.
weight	coefficient a_j of (free) variables in the knapsack inequality (coefficients a_j are not restricted to be nonnegative!).
idx	indices of the (free) variables in the knapsack inequality. If null it is assumed that variables are numbered from 0 to n-1.
xlp	LP solution of (free) variables appearing in the knapsack inequality.
rco	bsolute values of reduced costs of variables appearing in the knapsack inequality
lci	pointer to the generated cut pointer.

5.9.1.16 **VICcut**∗ VICsearchcut (**VICcut** ∗ cutlst, **VICcut** ∗ cut)

Searches for the cut to which the pointer "cut" is pointing to in a given list of cuts

Parameters

cutlst	pointer to the first cut in the linked list of cuts
cut	pointer to the cut which is search in the list

Returns

NULL if the cut is not contained in the list, and the pointer to the cut in the list of cuts otherwise

5.9.1.17 void VICsetdefaults ( )

Sets the above mentioned parameters to their above mentioned default values

5.9.1.18 void VICsort (int n, int ascending, int doinit, int size, void * numbers, int * order)

Sorting of arrays of integers/doubles

Parameters

n	dimension of the array that has to be sorted
ascending	sort in ascending (descending) order if ascending = 1 (0)
doinit	if doinit=1 it is assumed that the array "order" is not initialised
size	if size=sizeof(int) it is assumed that numbers points to integer array otherwise numbers must point to array of doubles
numbers	pointer to array of integers/doubles of dimension of at least n
order	on output the sorted array is given by numbers[order[0],...,order[n-1]]

5.9.1.19 void VICuflcov (CPXENVptr Env, int m, int n, char SolveCov, double * x, double * y, **VICcut** ** first_cut)

Tries to find violated combinatorial inequalities for the uncapacitated facility location problem. Let K be a subset of the set of all customers, and let J denote a subset of the set of potential depot sites. Define a binary matrix $a(i,j)$ for each $i \in K$ and $j \in J$. Let b denote (a lower bound on) the minimum number of depots $j \in J$ required to cover each customer $i \in K$, that is

$$b = \min \sum_{j \in J} y_j \tag{5.23}$$

$$s.t.: \sum_{j \in J} a_{ij} y_j \geq 1, \quad \forall i \in K \tag{5.24}$$

$$y_j = 0,1 \forall j \in J \tag{5.25}$$

Then the inequality $\sum_{i \in K} \sum_{j \in J} a_{ij} x_{ij} - \sum_{j \in J} y_j \leq |K| - b$ is valid for the UFLP. These inequalites generalize odd holes for the UFLP and have been proposed by D.C. Cho et al. (1983): "On the uncapacitated facility location problem I: Valid inequalities and facets", Mathematics of Operations Research 8, 579-589. See also G. Cornuejols, J.-M. Thizy (1982): "Some facets of the simple plant location polytope", Mathematical Programming 23, 50-74. Let $(x*, y*)$ denote a fractional solution. In order to find a violated inequality of the above type, the following simple heuristic is tried:

1. Set $J = \{j : 0 < y_j^* < 1\}$
2. Set $K = \{i : x_{ij}^* > 0 for more than one j \in J\}$
3. Solve the covering problem in order to find b (alternatively, find a lower bound on b by solving the LP relaxation of the covering problem and rounding up the objective function value)
4. Check if point $(x*, y*)$ violates the resulting inequality

Parameters

n	number of potential depot sites
m	number of customers
x	pointer to an array of doubles of size of at leat m∗n containing the allocation part of the fractional solution. Let $i = 0, ..., m-1$ and $j = 0, ...n-1$ be the indices of customers and depot sites, resp. Then x[i∗n + j] is the solution value of the allocation variable x(i,j), where 0 <= x(i,j) <= 1. The variable x(i,j) denotes the fraction of customer i's demand met from facility j.
y	pointer to an array of doubles of size of at least n containing the location part of the fractional solution, which should be separated by a submodular inequality. 0 <= y[j] <= 1, y[j]>=x[i,j]
first_cut	pointer to cut found by this procedure. The NULL pointer is returned if no violated inequality was found. See routine VICcflfc regarding definition of column indices.

5.9.1.20 void VICuflohi (int m, int n, double ∗ x, double ∗ y, **VICcut** ∗∗ first_ohi)

Generates odd-hole inequalities violated by the solution (x,y) for the UFLP. Several such cuts may be returned.

Parameters

n	number of potential depot sites
m	number of customers
x	pointer to an array of doubles of size of at leat m∗n containing the allocation part of the fractional solution. Let i=0,...,m-1 and j=0,...n-1 be the indices of customers and depot sites, resp. Then x[i∗n + j] is the solution value of the allocation variable x(i,j), where 0 <= x(i,j) <= 1. The variable x(i,j) denotes the fraction of customer i's demand met from facility j.
y	pointer to an array of doubles of size of at least n containing the location part of the fractional solution, which should be separated by a submodular inequality. 0 <= y[j] <= 1, y[j]>=x[i,j]
first_ohi	: pointer to the first cut of a linked list of violated odd-hole inequalities found by this procedure. The NULL pointer is returned if no violated inequality was found. See routine VICcflfc regarding definition of column indices.

```
VICcut* MyCutList = NULL;
VICcut* MyNewCut = NULL;

Solve_Something_like_the_LP_Relaxation_and_obtain_x_y;

VICuflohi( m, n, x, y, &MyNewCut );
if ( MyNewCut != NULL ) VICaddtolst( &MyCutList, MyNewCut );
```