ITR Project: comprehensive search implementation

Jie Xue

March 19, 2018

Contents

1 Introduction

1.1 Individual treatment recommendation

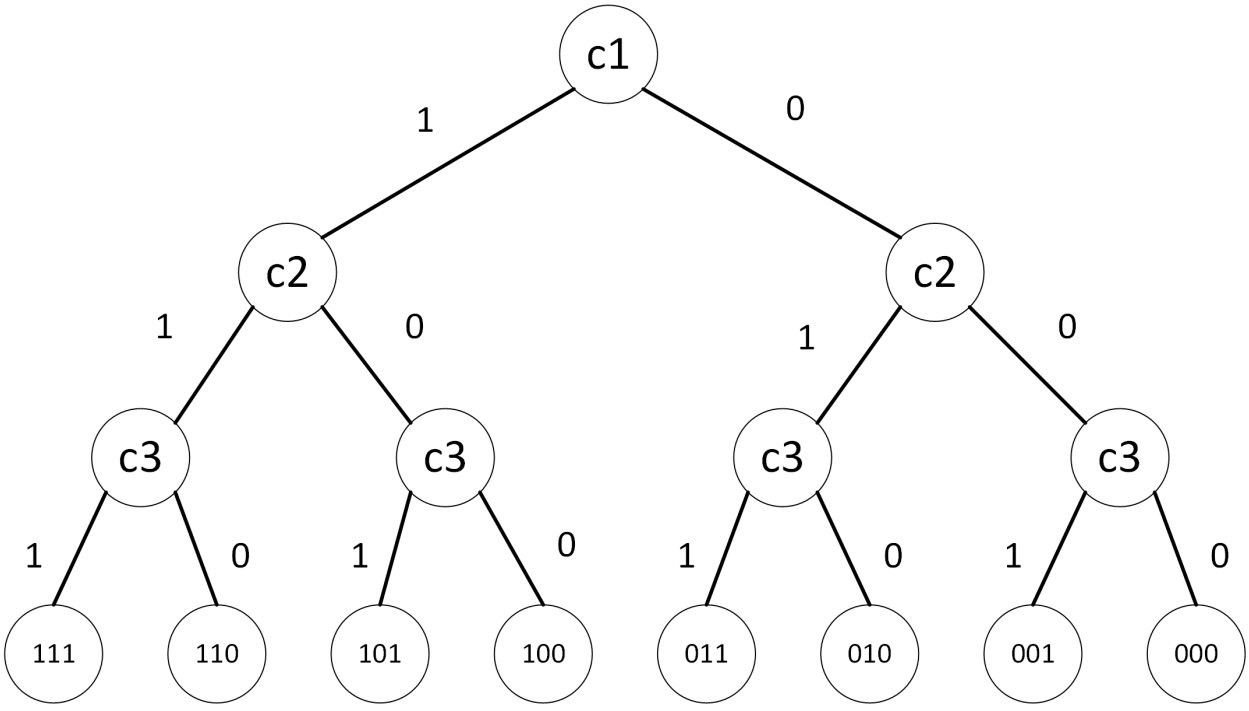
Accoding to Wiki, precision medicine is a medical model that proposes the customization of healthcare, with medical decisions, practices, and/or products being tailored to the individual patient. In this model, diagnostic testing is often employed for selecting appropriate and optimal therapies based on the context of a patient's genetic content or other molecular or cellular analysis. Tools employed in precision medicine can include molecular diagnostics, imaging, and analytics/software.

With new treatments and novel technology available, personalized medicine has become an important piece in the new era of medical product development. Traditional statistics methods for personalized medicine and subgroup identification primarily focus on single treatment or two arm randomized control trials. Motivated by the recent development of outcome weighted learning framework, we propose an alternative algorithm to search treatment assignments which has a connection with subgroup identification problems. Our method focuses on applications from clinical trials to generate easy to interpret results. This framework is able to handle two or more than two treatments from both randomized control trials and observational studies. We implement our algorithm in C++. Its performance is evaluated by simulations, and we apply our method to a dataset from a diabetes study.

1.2 Logic for comprehensive search

The comprehensive searching problem could be formulated as nested conditional operations with independent, exclusive mutual criteria.

Assume we have three criteria, c_1, c_2, and c_3. For each criteria, there would be a binary result for satisfied giving 1 or unsatisfied giving 0.



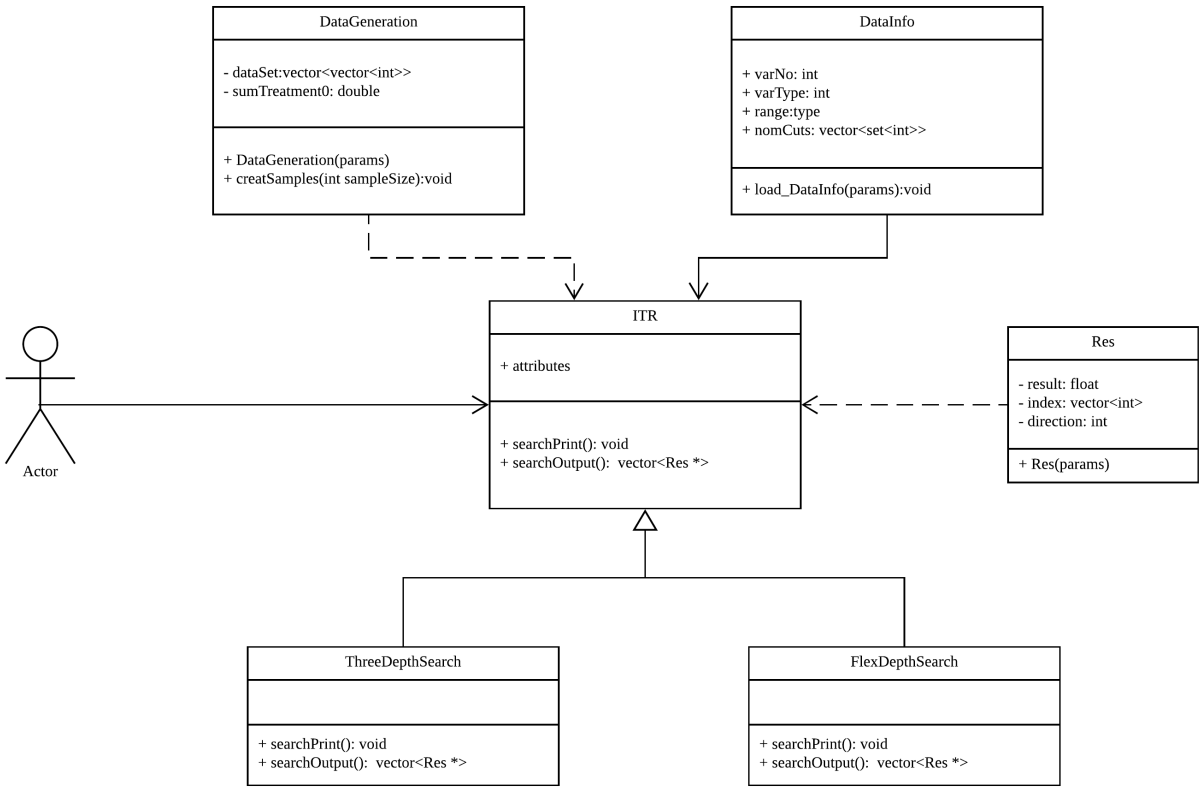
Therefore, there exists 8 possible results from 111 to 000 which fits a 8 length vector. The third criteria checks if the patient applies treatment 1. Then, the expectation for results 1 (11) use the information of 110, 100, 010, and 000. If we know the summation of all response that take treatment 0. Result 1 could be simplified as the function of 111 and 110.

2 Architecture and Design

2.1 Design goals

Implement ITR algorithm on C++. Apply appropriate design pattern with the consideration of abstraction, code reuse, stability, and scalability. The desired running time is less than 120s.

2.2 Class Diagram

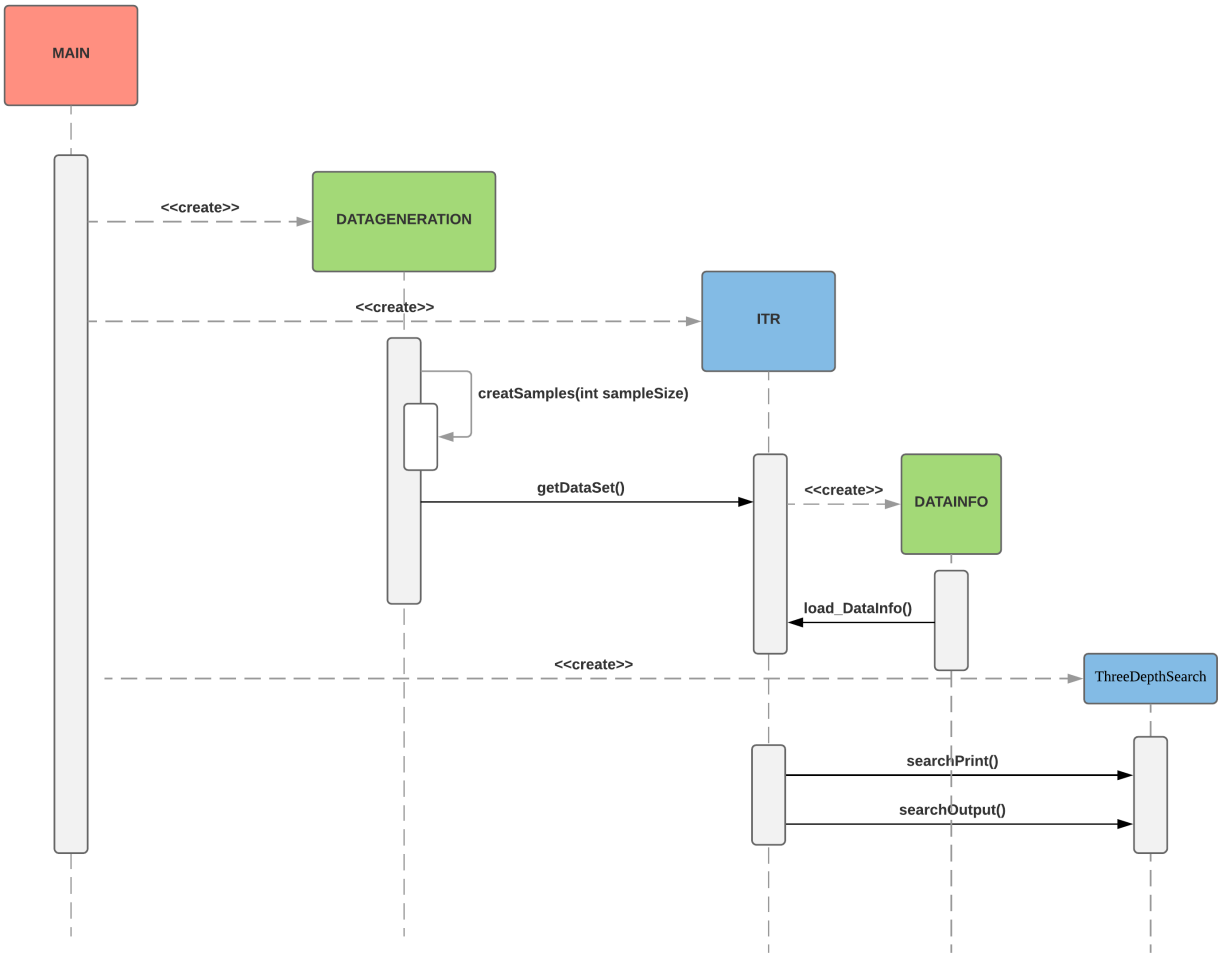


The ITR system is comprised of 4 main parts as following:

- DataGeneration: create raw simulation data
- DataInfo: generate searching information for each variable
- ITR: class for comprehensive search
  - ThreeDepthSearch: child class for ITR, fixed for depth = 3
  - FlexDepthSearch: child class for ITR, allow flexible depth input
- Res: class for store results

The template method design pattern is applied for ITR search.

2.3 Sequence Diagram

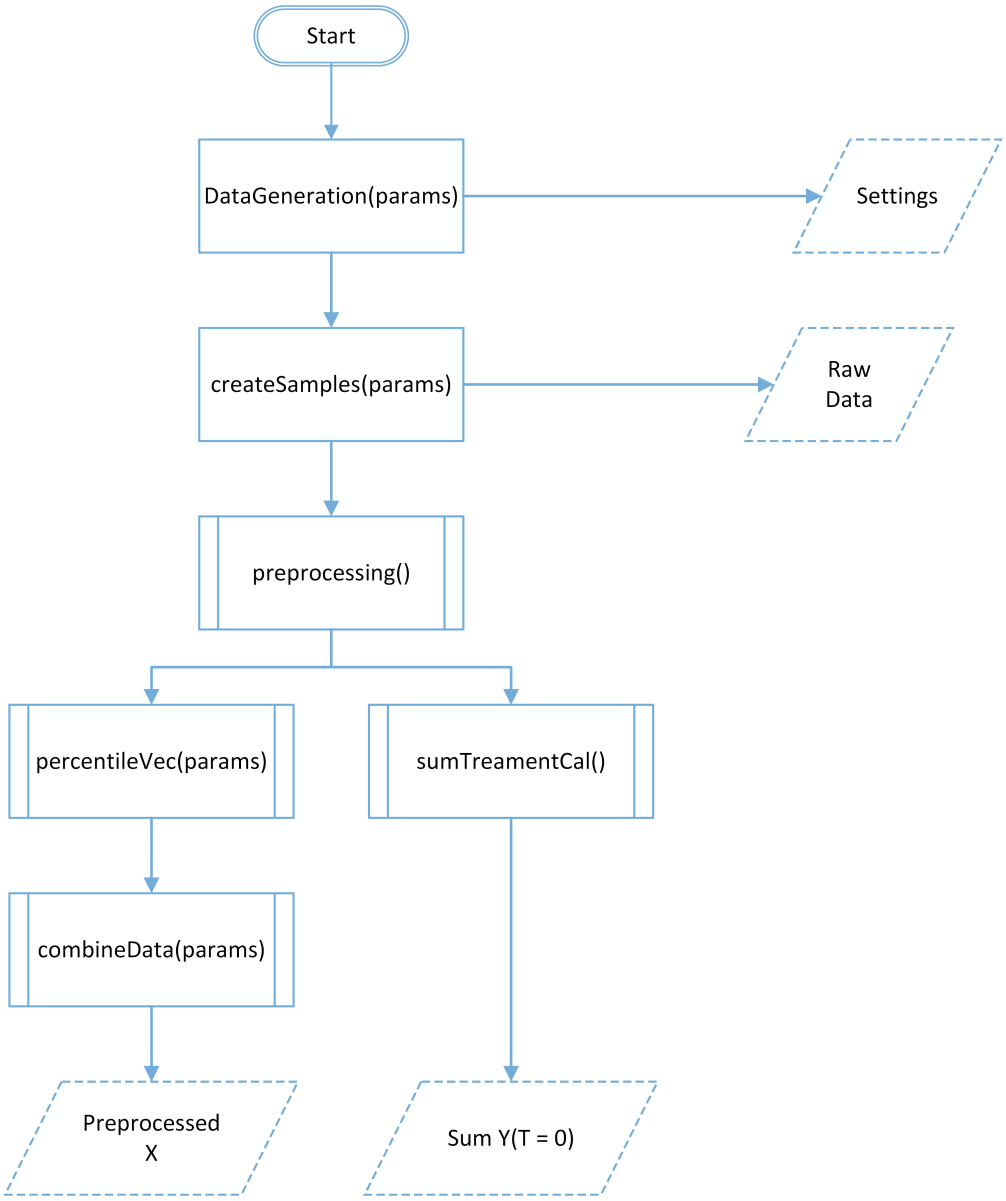


2.4 Detailed Class Design

2.4.1 DataGeneration Class Reference

The purpose of DataGeneration class is to create search ready data. Types of covariants and their corresponding ranges are given as settings through the constructor. Once the number of samples is given to the method "createSamples," the raw data will be generated. Then the "preprocessing" function will be called for percentile the continuous data (convert double to int) and sum the Y given Treatment equal to 0.

Flowchart of DataGeneration class:



Private Attributes:

- int sampleSize : no. of samples
- int covariateSize : no. of covariates
- vector<int> varType : vector for variable types, 0 for continuous, 1 for ordinal, 2 for nominal
- vector<double> rangesY : simulation range for Y
- vector<int> rangesActions : simulation range for Actions
- vector<double> rangesCont : simulation range for continuous variables
- vector<int> rangesOrd : simulation range for ordinal variables
- vector<int> rangesNom : simulation range for nominal variables

- vector<int> id : vector for patient ID
- vector<vector<double>> varY : 2D vector for response Y
- vector<vector<int>> actions : 2D vector for actions A
- vector<vector<double>> varCont : 2D vectors for continuous variables
- vector<vector<int>> varOrd : 2D vectors for ordinal variables
- vector<vector<int>> varNom : 2D vectors for nominal variables
- vector<vector<int>> varContInt : 2D vectors for percentile continuous variable
- vector<vector<int>> dataSet : 2D vectors for preprocessed variables
- double sumTreatment0 : sum of treatment 0

Public Member Function:

- **DataGeneration** (vector<int> const &vType, vector<double> const &rY, vector<int> const &rActions, vector<double> const &rCont, vector<int> const &rOrd, vector<int> const &rNom) : constructor
- void **createSamples** (int sSize) : create samples
- int **getSampleSize** () : return sample size
- int **getVariateSize** () : return covariate size
- int **getYSize** () : return response size
- int **getActionSize** () : return action size
- int **getContVarSize** () : return continuous variable size
- int **getOrdVarSize** () : return ordinal variable size
- int **getNomVarSize** () : return nominal variable size
- vector<int> **getVarType** () : return 1D variable type vector
- vector<vector<double>> **getY** () : return 2D response vector
- vector<vector<int>> **getActions** () : return 2D action vector
- vector<vector<double>> **getVarCont** () : return 2D continuous variable vector
- vector<vector<int>> **getVarCont** (int i) : return 2D percentile continuous variable vector
- vector<vector<int>> **getVarOrd** () : return 2D ordinal variable vector
- vector<vector<int>> **getVarNom** () : return 2D nominal variable vector
- vector<vector<int>> **getDataSet** () : return preprocessed data
- vector<int> **getID** () : return 1D patient vector
- double **getSumT0** () : return sum of treatment 0
- void **printY** () : print response vector
- void **printAction** () : print action vector
- void **printContVar** () : print continuous variable vector
- void **printContVar** (int i) : print percentile continuous variable vector

- void **printOrdVar** () : print ordinal variable vector
- void **printNomVar** () : print nominal variable vector
- void **printDataSet** () : print preprocessed data
- void **printInfo** (int i) : shortcut for print info.
- void **genReport** () : print all data info.

Private Member Function:

- void **preprocessing** ()
- vector<vector<int>> **combineData** (vector<vector<int>> const &varCont, vector<vector<int>> const &varOrd, vector<vector<int>> const &varNom)
- vector<int> **patientID** (int noSample)
- template<class T> int **getRangeSize** (vector<T> const &vectIn)
- vector<int> **createSeed** (int varsize, int start)
- vector<vector<int>> **assignSeed** (vector<int> const &seed, vector<int> const &varType)
- vector<double> **dataGenerator** (int seed, double lowerBound, double upperBound)
- vector<int> **dataGenerator** (int seed, int lowerBound, int upperBound)
- template<class T> vector<vector<T>> **sampleGenerator** (vector<int> const &seed, vector<T> const &ranges)
- vector<int> **percentileVec** (vector<double> const &vectIn)
- vector<vector<int>> **percentileVec** (vector<vector<double>> const &vectIn)
- map<double, int> **percentileMap** (vector<double> const &vectorIn)
- double **percentile** (double len,double index)
- int **assignPercentile** (double p)
- void **sumTreatmentCal** ()
- template<class T> void **print1DVector** (vector<T> const &vectIn)
- template<class T> void **print2DVector** (vector<vector<T>> const &vectIn)

More information about this class can be found from the following files:

- DataGeneration.h
- DataGeneration.cpp

2.4.2 DataInfo Class Reference

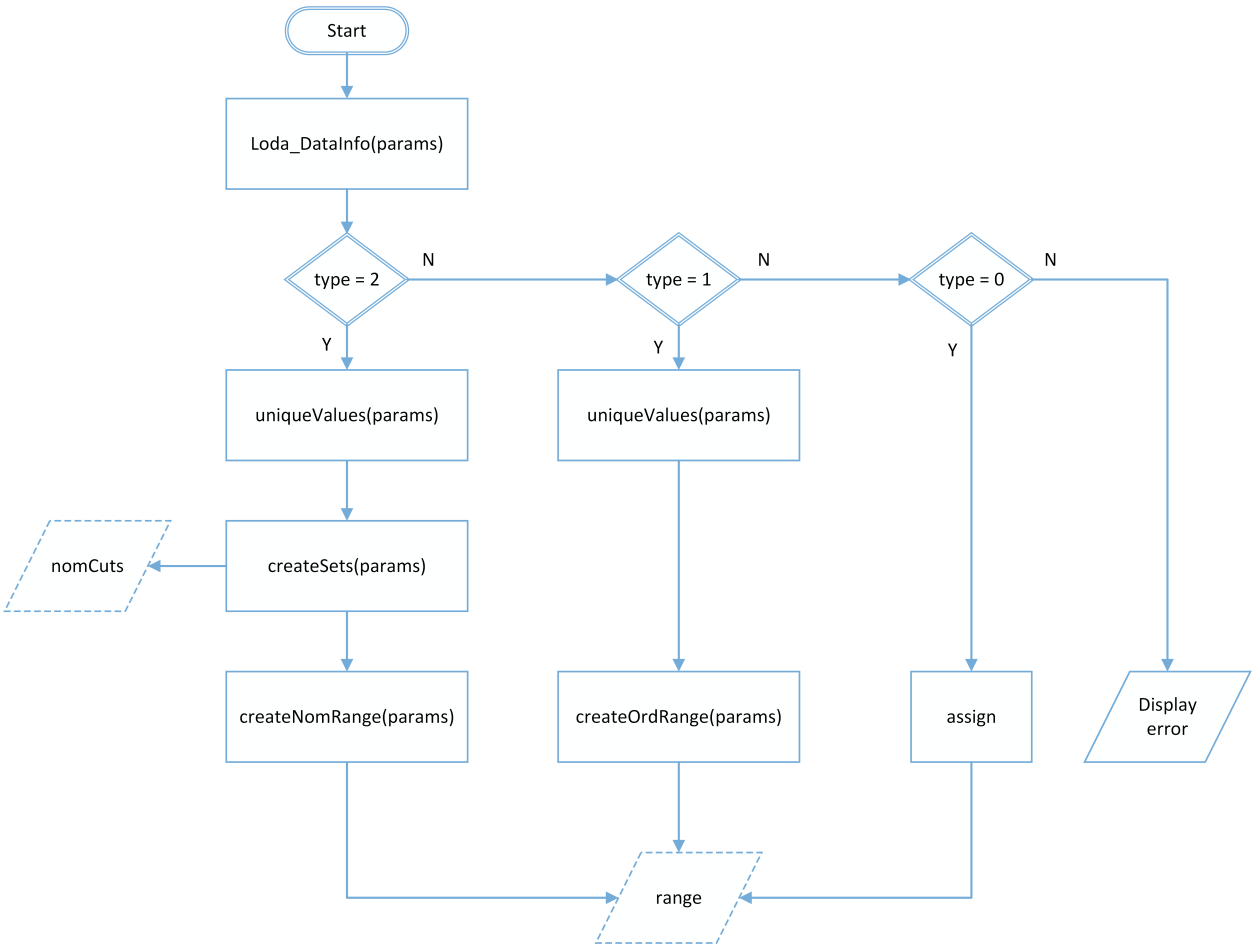
The purpose of DataInfo class is generating the searching information for each covariant.

- Continuous variable:
  - Assign cut range $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ for available number $x \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Ordinal variable:
  - Get unique value and find the corresponding cut range

  - Eg: Cut range $[1, 2, 3, 4, 5]$ for $x \in \{0, 1, 2, 3, 4\}$

- Nominal variable:
  - Get unique value and find the corresponding cuts and cut range
  - Eg: For $x \in \{0, 1, 2\}$, all possible subsets are $\{\}, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\}$. However, we only choose the short ones. As a result, inside the "nomCuts" we have $\{\}, \{0\}, \{1\}, \{2\}$.

Flowchart of DataInfo class:



Private Attributes:

- int varNo : variable no.
- int varType : variable type, 0 for continuous, 1 for ordinal, 2 for nominal
- vector<int> range : vector for store cut range
- vector<set<int>> nomCuts : vector for store nominal cuts

Public Member Functions

- void **load_DataInfo** (int no, int type, vector<int> &dataSetColumn) : method works as constructor
- int **getVarNo** () : return variable no.

- int **getVarType** () : return variable type
- int **getCutSize** () : return no. of cuts
- bool **nomContains** (int x, int index) : return if nomCuts[index] contain x
- int **getRange** (int i) : return range no. i
- void **printVarInfo** () : print all variable info. in each object
- void **printSet** (int i) : print set i

Private Member Functions

- vector<int> **uniqueValues** (vector<int> &vectorIn)
- vector<set<int>> **createSets** (vector<int> vectorIn)
- vector<int> **createNomRange** (int i) : create cut range for nominal variable
- vector<int> **createOrdRange** (vector<int> x) : create cut range for percentile continuous/ordinal variable

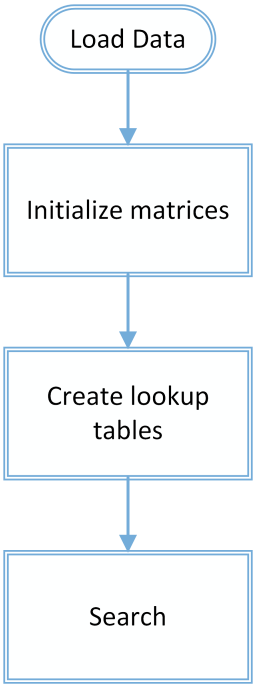
More information about this class can be found from the following files:

- DataInfo.h
- DataInfo.cpp

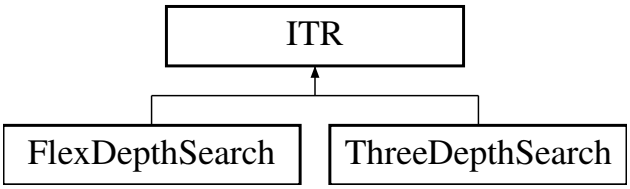
2.4.3 ITR Class Reference

The main purpose of Abstract class is create the binary lookup table and virtual searching functions.

Flowchart for ITR class:



Inheritance diagram for ITR:



Public Member Functions

- **ITR** (DataGeneration &data, int d): constructor, d for search depth
- int **getSampleSize** ()
- int **getVarSize** ()
- int **getActionSize** ()
- int **getYSize** ()
- int **get_CutSize** ()
- int **get_X** ()
- int **get_Action** ()
- double **get_Y** ()
- bool **get_X_Table** ()
- void **print_X** ()
- void **print_Action** ()
- void **print_Y** ()
- void **print_All** ()
- void **print_Range** ()
- void **print_Type** ()
- void **print_CutTable** ()
- void **print_VarInfo** (int i)
- void **print_VarInfo** ()
- void **print_Combinations** ()
- virtual void **searchPrint** () : virtual function for print comprehensive search results
- virtual vector<Res *> **searchOutput** () : virtual function for store comprehensive search results

Protected Attributes

- vector<int> var_Type : vector for variable type
- int depth: depth
- vector<vector<int>> lookup : lookup table
- int** var_X : 2D matrix for X (sample_Size × var_Size)

- int** var_A : 2D matrix for A (sample_Size × action_Size)
- double** var_Y : 2D matrix for A (sample_Size × y_Size)
- bool*** table_X : 3D lookup table (var_Size × cut_Size[i] × sample_Size)
- DataInfo* info: 1D variable info. vector (var_Size)
- int* cut_Size: 1D cut size vector (var_Size)
- int sample_Size : sample size
- int var_Size : variable size
- int action_Size : action size
- int y_Size : response size
- double T0 : sum of treatment 0

Private Member Functions

- void **init** ()
- void **init_TableX** ()
- void **load_CutSize** ()
- void **load_table_X** (vector<vector<int>> x)
- void **load_X** (vector<vector<int>> x)
- void **load_Action** (vector<vector<int>> a)
- void **load_Y** (vector<vector<double>> y)
- void **cleanAll** ()
- vector<vector<int>> **combine** (int n, int k)

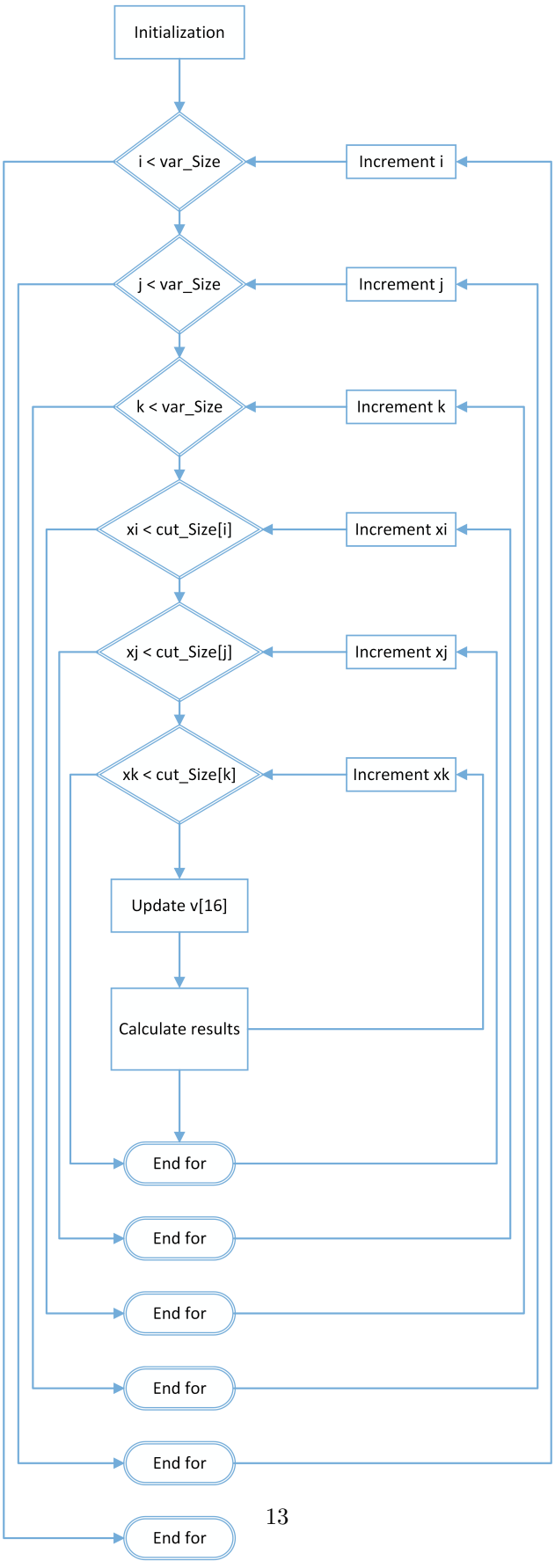
More information about this class can be found from the following files:

- ITR.h
- ITR.cpp

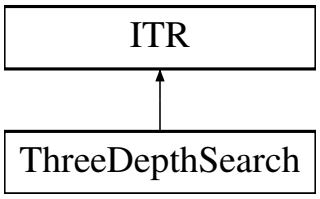
2.4.4 ThreeDepthSearch Class Reference

The purpose of ThreeDepthSearch class is for 3 depth comprehensive search.

Flowchart for ThreeDepthSearch class:


Initialization
i < var_Size
Increment i
j < var_Size
Increment j
k < var_Size
Increment k
xi < cut_Size[i]
Increment xi
xj < cut_Size[j]
Increment xj
xk < cut_Size[k]
Increment xk
Update v[16]
Calculate results
End for
End for
End for
End for
End for
End for

Inheritance diagram for ThreeDepthSearch:



Public Member Functions

- **ThreeDepthSearch** (DataGeneration &data, int d) : call parent constructor
- void **searchPrint** () : override function
- vector<Res* > **searchOutput** () : override function

More information about this class can be found from the following files:

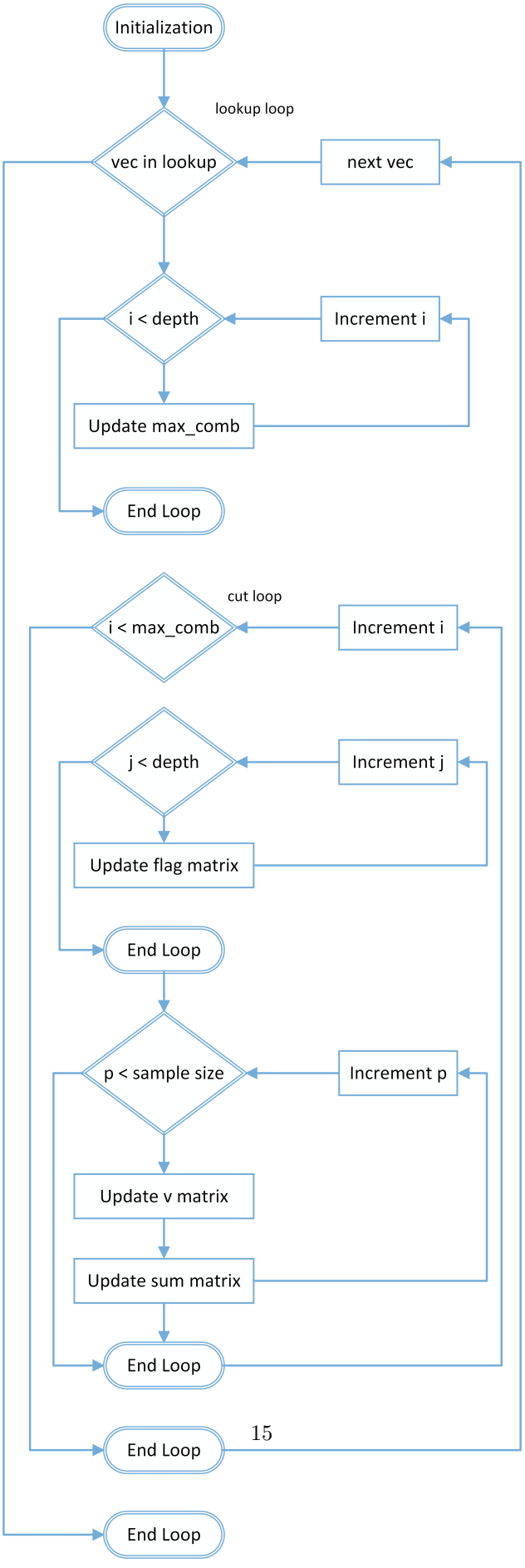
- ThreeDepthSearch.h
- ThreeDepthSearch.cpp

2.4.5 FlexDepthSearch Class Reference

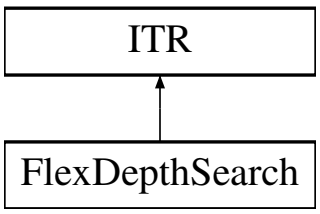
The purpose of FlexDepthSearch class is for flexible depth comprehensive search. It is mainly comprised of three loops:

- Outer loop: loop through all variable combinations
- Middle loop: loop through all possible cuts for given variable(s)
- Inner loop: loop through samples for searching

Flowchart for FlexDepthSearch class:


Initialization
lookup loop
vec in lookup
next vec
i < depth
Increment i
Update max_comb
End Loop
cut loop
i < max_comb
Increment i
j < depth
Increment j
Update flag matrix
End Loop
p < sample size
Increment p
Update v matrix
Update sum matrix
End Loop
End Loop
End Loop

Inheritance diagram for FlexDepthSearch:



Public Member Functions

- **FlexDepthSearch** (DataGeneration &data, int d) : call parent constructor
- void **searchPrint** () : override function
- vector<Res* > **searchOutput** () : override function

More information about this class can be found from the following files:

- FlexDepthSearch.h
- FlexDepthSearch.cpp

2.4.6 Res Class Reference

Private Attributes:

- float result: results
- vector<int>* index: variable combination information
- int direction: cut direction information

Public Member Functions

- **Res** (double res, vector<int> ind, int dir): construction
- float **getRes** (): return results
- vector<int>* **getIndex** (): return variable combination
- int **getDirection** (): return cut directions

More information about this class can be found from the following files:

- Res.h
- Res.cpp

3 Test

In this simulation, we proposed 3000 samples, each with total 35 variables. The 35 variables are comprised of 25 continuous variables, 5 ordinal variables (range 0 to 4), and 5 nominal variables (range 0 to 4). A binary treatment is assigned uniformly for each patient. The response is assigned randomly to the range of 0 to 100.

3.1 System information

- System Type: System Type x64-based PC
- Processor: Processor Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz, 2401 Mhz, 2 Core(s), 4 Logical Processor(s)
- Installed Physical Memory (RAM): 8.00 GB

3.2 Run-time performance test

- Fixed step search (ThreeDepthSearch)
  - Print: around 60s
  - Output: around 50s
- Flexible step search (FlexDepthSearch) with depth = 3
  - Print: around 90s
  - Output: around 75s

3.3 Unit testing

Store 100 samples, 9 variables(3 continuous variables, 3 continuous variables, 3 nominal variables)

1. Data Input Test
   (a) Test correct input for class "LoadSimpleTestData"
   (b) Test correct input data size for class "LoadCSVData"
   (c) Test correct input data value for class "LoadCSVData"
   (d) Test correct data size for class "CreateSimulationData"
2. Preprocessing Test
   (a) Test class "LoadData"
      i. Test load simple test data
      ii. Test load CSV data
      iii. Test load simulation data
   (b) Test class "CreateX"
      i. Test table X by using loaded simple test data

ii. Test table X by using loaded CSV data

iii. Test table X by using loaded simulation data

(c) Test class "CreateLookUpTable"

i. Test load simple test data

ii. Test load CSV data

iii. Test load simulation data

3. Search Test

(a) Test class "ThreeDepthSearch"

(b) Test class "FlexDepthSearch"

4 Future work

1. Add correctness test modular for 1, 2, 3 depth of comprehensive search (100 samples)
2. Add more recommendation algorithms
3. Add more data preprocessing methods, eg. data cleaning method
4. May apply strategy pattern for better flexibility
5. Code discussion with Michael A Bell ¡bell_michael_a@lilly.com¿