Object subclass: #Optimization
  instanceVariableNames: 'minimizeFlag quaIC quaICString'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'srOptimization'

!Optimization methodsFor: 'accessing'
minimizeFlag
  "minimizeFlag!
minimizeFlag: aBoolean
  minimizeFlag := aBoolean!
quaIBC
  "quaIBC!
quaIBC: aBC
  quaIBC := aBC!
quaIBCString
  "quaIBCString!
quaiBString: aString
  quaIBCString := aString!
!
!Optimization methodsFor: 'initialize'
initialize
  quaIBCString notNil & quaIBC isNil1 ifTrue: [quaIBC := Compiler evaluate: quaIBCString]!
!
Optimization subclass: #PopulationBasedOptimization
  instanceVariableNames: 'generation population popSize fittestAtAll randNo randSeed random evaluations'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'srOptimization'

!PopulationBasedOptimization methodsFor: 'accessing'
evaluations
  "evaluations!
fittestAtAll
  "fittestAtAll!
fittestAtAll: aFittest
  fittestAtAll := aFittest.
  self changed: #fittestAtAll!
geneneration
  "generation!
popSize
  "popSize!
popSize: aPositiveInteger
  popSize := aPositiveInteger!
population
  "population!
randNo
  "randNo!
randNo: aValue
  randNo := aValue!
randSeed
  "randSeed!
randSeed: aValue
  randSeed := aValue!
!
!PopulationBasedOptimization methodsFor: 'user interface'
nextGeneration
  "self subclassResponsibility!
nextGenerations: numOfGenerations
  nextGenerations timesRepeat: [self nextGeneration]!
start
  "generation = 0 ifTrue: [self nextGenerations: self generations]!!

PopulationBasedOptimization subclass: #PBVectors
  instanceVariableNames: 'vectorSize'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'srOptimization'

!PBVectors methodsFor: 'accessing'
vectorSize
  "vectorSize!
vectorSize: aPositiveInteger
  vectorSize := aPositiveInteger!

PBVectors subclass: #PNNumbers
  instanceVariableNames: 'fromVector toVector'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'srOptimization'!
fromVector := aVector!

toVector := aVector!

PBNNumbers subclass: #SREvol
instanceVariableNames: 'stochasticVector numOfBest moveMethod mueMoveFactor sigmaReduceFactor
classVariableNames: '

category: 'srOptimization'!

SREvol methodsFor: 'initialize-release'
initialize
| vec medianFittestVec |
| vec := Vector new: vectorSize withAll: 0.
| 1 to: numOfBest do: [:ix | vec := vec + (population at: ix) vector].
| medianFittestVec := vec / numOfBest.
| moveMethod := #null
| ifTrue: [stochasticVector
| moveTo: medianFittestVec
| mueMoveFactor: mueMoveFactor
| sigmaFactor: sigmaReduceFactor]
| ifFalse: [moveMethod = #sub
| ifTrue: [stochasticVector
| moveTo: medianFittestVec
| mueMoveFactor: mueMoveFactor
| sigmaSubtrahentVector: sigmaSubtrahentVector]
| ifFalse: [self error: 'moveMethod unknown']]

SREvol methodsFor: 'accessing'

evaluations := evaluations + 1

numOfBest

mueMoveFactor :=mueMoveFactor

mueMoveFactor := mueMoveFactor

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue!

mueMoveFactor := aValue!

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue

mueMoveFactor := aValue
sigmaTarget := aNum!
sigmaTarget := aNum!

stochasticVector

"stochasticVector! !

\[ \text{medianUpdateProbabilityVector} \]

medianUpdateProbabilityVector

"first compute median vector of them in best; second move in direction of it"

| vec |
| vec := Vector new: vectorSize withAll: 0.
| 1 to: numberOfBest do: [:ix | vec := vec + (population at: ix) vector].
| probabilityVector := probabilityVector + (vec / numberOfBest - probabilityVector * learningRate)]

mutateProbabilityVector

| prob |
| 1 to: VectorSize do: [:ix | random next < mutateProbability]
| ifTrue: [prob := probabilityVector at: ix.
| probabilityVector at: ix put: prob + ((random next < 0.5 ifTrue: [0] ifFalse: [0]) - prob * mutateShift)].]

singleUpdateProbabilityVector

"first moving probability vector by the worse vectors in the best set and at last by the best vector of all is better than the opposite, because we move with learningRate in direction of actual vector in best set and should at last make a move in the best found direction!"

| vec |
| vec := (population at: ix) vector.
| probabilityVector := probabilityVector + (vec - probabilityVector * learningRate)]

updateProbabilityVector

updateProbabilityVectorMethod := #median ifTrue: ['self medianUpdateProbabilityVector'].
updateProbabilityVectorMethod := #single ifTrue: ['self singleUpdateProbabilityVector'].

self error: 'unknown updateProbabilityVectorMethod' !

PBILGenerator methodsFor: 'action'!

nextGeneration

self generateSamples.
self updateFittestAtAll.
self updateProbabilityVector.
self mutateProbabilityVector.

generation := generation + 1.
self changed: #nextGeneration! !

PBILGenerator methodsFor: 'accessing'!

learningRate

"learningRate!" learningRate := aPositiveNum
learningRate := aPositiveNum!

mutProbability

"mutProbability!"

mutProbability := aPositiveNum
mutProbability := aPositiveNum!

mutShift

"mutShift!"

mutShift := aPositiveNum
mutShift := aPositiveNum!
Smalltalk-Sourcen

`numOfBest!`  
`numOfBest := aPositiveInteger!`  
`probabilityVector`  
`"probabilityVector!`  
`updateProbabilityVectorMethod`  
`"updateProbabilityVectorMethod!`  
`updateProbabilityVectorMethod: aSymbol`  
`updateProbabilityVectorMethod := aSymbol!`  

`PBILGeneral methodsFor: 'initialize'!`  
`initialize`  
`super initialize.`  
`probabilityVector := Vector new: vectorSize withAll: 0.5!`  
`^probabilityVector!`  

`PBILGeneral class`  
`instanceVariableNames: '!'`  

`PBILGeneral class methodsFor: 'instance creation'!`  
`minimizeFlag: mBool quaIC: qBlockClosure popSize: psInt vectorSize: vsInt learningRate: lrNum mutProbability: mNum mutShift: mNum randNo: aNo randSeed: aSeed`  
`super`  

`PBIL class`  
`instanceVariableNames: '!'`  

`PBIL class methodsFor: 'instance creation'!`  

`PBIL class`  
`instanceVariableNames: '!'`  

`PBIL class methodsFor: 'instance creation'!`  

`ApplicationModel subclass: #OptimizationTool`  
`instanceVariableNames: 'decodeVecIC indices optimization codedMultiDataDict multiDataDict functionToBeCoded computingFlag generationAsp mutationAspVectorAsp evaluationAsp functionAsp generationAsp learningRateAsp upvmSymbol mutProbabilityAsp mutShiftAsp randomAsp'`  

`PBILGeneral subclass: #PBIL`  
`instanceVariableNames: '!'`  
`category: 'srOptimization'!`
functionToBeCoded := self functionTypeAsp value = '#16BitsToNum' | (self functionTypeAsp value = '#32BitsToNum').
self updateCodingVectorComps!

generationsChanged
  "generations * popSize = evaluations"
computingFlag ifTrue: ['self'].
computingFlag := true.
self evaluationsAsp value > 0 ifTrue: [self popSizeAsp value = self evaluationsAsp value // self generationsAsp value] !

ifFalse: [self popSizeAsp value > 0 ifTrue: [self evaluationsAsp value = self popSizeAsp value * self evaluationsAsp value] // self generationsAsp value].
sigmaTargetAsp value > 0 & (self generationsAsp value > 0) ifTrue: [self sigmaReduceFactorAsp value = self sigmaTargetAsp value ** (1 / self generationsAsp value)].
computingFlag := false!

moveMethodChanged
  | comps |
  comps := #(!#sigmaReduceFactor #sigmaReduce ).
moveMethodAsp value = #0 !|
moveMethodAsp value = #sub !|
moveMethodAsp value = #sqrt !|
moveMethodAsp value = #sub !| (comp do: [:comp | (self builder componentAt: comp) enable])

optTypeChanged
  | optType |
  optType := self optTypeAsp value.
self parameterComponents do: [:comp | (self builder componentAt: comp) disable].

updateCodeDict
  "after decoding from Bits to Numbers no further reducing"

1. vec
vecCodeMultiDataDict isNotEmpty not ifFalse: [vec := self codeMultiDataDict]
1. vec
vecCodeMultiDataDict isNotEmpty not ifFalse: [vec := self codeMultiDataDict]
1. vec
vecCodeMultiDataDict isNotEmpty not ifFalse: [vec := self codeMultiDataDict]
Smalltalk-Sourcen

keysAndValuesDo:
{[key : val |
  key = #fittestAtAllQualPlotAsp
  ifTrue: [num := optimization fittestAtAll quality]
  ifFalse: [key = #bestOffPopQualPlotAsp]
  ifTrue: [num := optimization bestOfPopulation quality]
  ifFalse: [key = #worstOffPopQualPlotAsp]
  ifTrue: [num := optimization worstOfPopulation quality]
  ifFalse: [self error: 'don’t know aspect']][].
val addLast: optimization generation $ num]])
updateDataDicts
  self plotsAsp value = $ no
  ifTrue:
    [self updateDataDict.
     self updateMultiDataDict.
     self updateCodeMultiDataDict]!!

updateFittestAtAllVecValueHolder
fittestAtAllVecValueHolder value: {self functionToBeCoded}
  ifTrue: [decodeVecBC value: optimization fittestAtAll vector]
  ifFalse: [optimization fittestAtAll vector]]
updateMultiDataDict
"reduced data through 'indexing'"
| vec |
multiDataDict isNil not ifTrue: [multiDataDict
  keysAndValuesDo:
  {[key : val |
    key = #probabilityVecPlotAsp
    ifTrue: [vec := optimization probabilityVector]
    ifFalse: [key = #fittestAtAllVecPlotAsp]
    ifTrue: [vec := optimization fittestAtAll vector]
    ifFalse: [key = #bestOffPopVecPlotAsp]
    ifTrue: [vec := optimization bestOfPopulation vector]
    ifFalse: [key = #worstOffPopVecPlotAsp]
    ifTrue: [vec := optimization worstOfPopulation vector]
    ifFalse: [key = #meanVecPlotAsp]
    iftrue: [vec := optimization stochasticVector means]
    ifFalse: [key = #sigmaVecPlotAsp]
    ifTrue: [vec := optimization stochasticVector sigmas]
    ifFalse: [self error: 'wrong aspect value']]].
val addLast: optimization generation $ (vec vectorFromIndices: indices)]]]
updateMultiDataDict
"reduced data through 'indexing'"
| vec |
multiPlotDict isNil not ifTrue: ['self'.
multiPlotDict isEmpty not ifTrue: [multiPlotDict
  keysAndValuesDo:
  {[key : val |
    key = #probabilityVecPlotAsp
    ifTrue: [vec := optimization probabilityVector]
    ifFalse: [key = #fittestAtAllVecPlotAsp]
    ifTrue: [vec := optimization fittestAtAll vector]
    ifFalse: [key = #bestOffPopVecPlotAsp]
    ifTrue: [vec := optimization bestOfPopulation vector]
    ifFalse: [key = #worstOffPopVecPlotAsp]
    ifTrue: [vec := optimization worstOfPopulation vector]
    ifFalse: [key = #meanVecPlotAsp]
    iftrue: [vec := optimization stochasticVector means]
    ifFalse: [key = #sigmaVecPlotAsp]
    ifTrue: [vec := optimization stochasticVector sigmas]
    ifFalse: [self error: 'don’t know aspect']]].
val addLast: optimization generation $ (vec vectorFromIndices: indices).
self everyFew ifTrue: [key = #probabilityVecPlotAsp ifTrue: [val getPointYRanges]]]!!
updatePlotDicts
  self plotsAsp value = $ no
  ifTrue:
    [self updatePlotDict.
     self updateMultiPlotDict.
     self updateCodeMultiPlotDict]!!

!OptimizationTool methodsFor: 'building'!
postBuildWith: aBuilder
  (aBuilder componentAt: #reechenberg) disable.
  self optTypeAsp value: #pbiGeneral.
  self functionTypeAsp value: #onBits.
  self plotsAsp value: #no.
  self computingAsp value: #foreground.
  self minimizeAsp value: #true.
  self moveMethodValue: #null.
postOpenWith: aBuilder
  "request notification about the close event."
super postOpenWith: aBuilder.
  super windowApplication: self.
  super window sendWindowEvents: [#Close]!!

!OptimizationTool methodsFor: 'initialize--release'!
initialize
  self optTypeAsp onChangeSend: optTypeChanged to: self.
  self functionTypeAsp onChangeSend: functionTypeChanged to: self.
  self evaluationsAsp onChangeSend: #evaluationsChanged to: self.
  self generationsAsp onChangeSend: #generationsChanged to: self.
  self popSizeAsp onChangeSend: #popSizeChanged to: self.
  self computingFlag := false.
  self functionToBeCoded := false.
  self sigmaReduceFactorAsp onChangeSend: #sigmaReduceFactorChanged to: self.
  self sigmaTargetAsp onChangeSend: #sigmaTargetChanged to: self.
  self moveMethodAsp onChangeSend: #moveMethodChanged to: self.
  fittestAtAllVecValueHolder := nil asValue!
release
super release.
self optimization removeDependent: self!!

!OptimizationTool methodsFor: 'actions'!!
initOptimization
  | opt minimizeFlag qBC | minimizeFlag := self minimizeFlag value = #true
  | true.
  ifFalse: [true] ifFalse: [false]
  ifFalse: [self minimizeAsp value = #false]
  ifTrue: [false]
  ifFalse: [self error: 'wrong aspect value']]
qBC := Compiler evaluate: self functionAsp value.
self functionToBeCoded
ifTrue:
  [qualBC := self codeFunc: qBC.
   self setDecodeVecBC].
ifFalse: [qualBC := qBC].

self optTypeAsp value = #pbl ifTrue: [opt := PBL
  minimizeFlag: minimizeFlag
  qualBC := qualBC
  popSize: self popSizeAsp value
  vectorSize: self vecSizeAsp value
  learningRate: self learningRateAsp value
  mutProbability: self mutProbabilityAsp value
  mutShift: self mutShiftAsp value
  randBG: self randNoAsp value
  randSeed: self randSeedAsp value].

self optTypeAsp value = #pblGeneral ifTrue: [opt := PBLGeneral
  minimizeFlag: minimizeFlag
  qualBC := qualBC
  popSize: self popSizeAsp value
  vectorSize: self vecSizeAsp value
  numOfBest: self numOfBestAsp value
  learningRate: self learningRateAsp value
  updateProbabilityVectorMethod: self updateMethodAsp value
  mutProbability: self mutProbabilityAsp value
  mutShift: self mutShiftAsp value
  randBG: self randNoAsp value
  randSeed: self randSeedAsp value].

self optTypeAsp value = #mc ifTrue: [opt := MCG
  minimizeFlag: minimizeFlag
  qualBC := qualBC
  popSize: self popSizeAsp value
  vectorSize: self vecSizeAsp value
  from: (Compiler evaluate: self fromVectorAsp value)
  to: (Compiler evaluate: self toVectorAsp value)
  numOfBest: self numOfBestAsp value
  moveToFactor: self moveToFactorAsp value
  stochasticMoveMethod: self moveMethodAsp value
  randBG: self randNoAsp value
  randSeed: self randSeedAsp value].

self optimization: opt!

inspectFittestAtAll

"This stub method was generated by UIDefiner"

"self optimization method inspect"

ifFalse: [self]!!

inspectOpt

"This stub method was generated by UIDefiner"

"self optimization method inspect"

ifFalse: [self]!!

inspectVecDecoded

((decodeVecBC value: optimization fittestAtAll vector)
  collect: [num | num asFloat]) inspect!}

Bindings notNil ifTrue: [Bindings keysAndValuesDo: [:symbol :asp | (**Asp** match: symbol asString)
  ifTrue: [(self perform: symbol)
    value: asp] value]]

openPlots

[| bigMultiPlotDict |
  plotDict isEmpty not ifTrue: [plotDict keysAndValuesDo: [:label :plotModel | (SinglePlot
  view model: plotModel)
    openWithLabel: label]].

bigMultiPlotDict := Dictionary new.

multiPlotDict associationsDo: [:ass | bigMultiPlotDict add: ass].

codedMultiPlotDict associationsDo: [:ass | bigMultiPlotDict add: ass].

bigMultiPlotDict isEmpty not ifTrue: [bigMultiPlotDict keysAndValuesDo: [:label multiPlotModel |
  (MultiPlotView model: multiPlotModel)
    openWithLabel: label]].

openPlotAtEnd

[bigMultiDataDict |
  self plotsAsp = #false ifFalse: ['**self**.
  dataDict isEmpty not ifTrue: [dataDict keysAndValuesDo: [:label :points | (SinglePlotView
  model: ((SinglePlotModel points: points)
    plotStyle: #lines))
    openWithLabel: label]].

bigMultiChartData := Dictionary new.

multiChartData associationsDo: [:ass | bigMultiChartData add: ass].

bigMultiChartData isEmpty not ifTrue: [bigMultiChartData keysAndValuesDo: [:label :yVecCol
  1 | (MultiPlotView model: ((MultiPlotModel yVecPoints: yVecCol)
    setPlotsStyle: #lines))
    openWithLabel: label]].]

saveControls

**

Bindings := Dictionary new.

self builder bindings keysAndValuesDo: [:k :v | Bindings at: k put: v copy]!!

start

**

| actionBlock |
  self indicesNecessary
  ifTrue: [indices := Compiler evaluate: self indicesAsp value.
    indices isNil ifTrue: ["Dialog warn: 'indices for plots necessary!!'"]].
  actionBlock := [
    [indices := Compiler evaluate: self indicesAsp value.
      indices isNil ifTrue: ["Dialog warn: 'indices for plots necessary!!'"].
      actionBlock := [
        [indices := Compiler evaluate: self indicesAsp value.
          indices isNil ifTrue: ["Dialog warn: 'indices for plots necessary!!'"].
          actionBlock := [self initPlotDicts].
        ]
      ]
    ]
  ]

start2

**

| actionBlock |
  self indicesNecessary
  ifTrue: [indices := Compiler evaluate: self indicesAsp value.
    indices isNil ifTrue: ["Dialog warn: 'indices for plots necessary!!'"].
    actionBlock := [self initPlotDicts]}.
self initOptimization.
self plotsAsp value = #begin ifTrue: [self openPlots].
self computingAsp value = #end ifFalse: [self openPlots]].
self computingAsp value = #foreground
ifTrue: [actionBlock value]
ifFalse: [UserProcessManager actualManager startBlock: actionBlock]
startData
**
| actionBlock |
| self indicesNecessary
ifTrue: [indices := Compiler evaluate: self indicesAsp value.
indices isNil ifTrue: ['Dialog warn: 'indices for plots necessary!!'].]
actionBlock :=
| self initDataDicts.
| self initOptimization.
| self startOptimization.
| self openPlotsAtEnd].
self computingAsp value = #foreground
ifTrue: [actionBlock value]
ifFalse: [UserProcessManager actualManager startBlock: actionBlock]
startOptimization
self optimization nextGenerations: self generationsAsp value.
Transcript cr; print: self optimization fitTestAtAll1; flush!
startPlot
**
| actionBlock |
| self indicesNecessary
ifTrue: [indices := Compiler evaluate: self indicesAsp value.
indices isNil ifTrue: ['Dialog warn: 'indices for plots necessary!!'].]
actionBlock :=
| self initDataDicts.
| self initOptimization.
| self computingAsp value = #begin ifTrue: [self openPlots].
| self startOptimization.
| self computingAsp value = #end ifFalse: [self openPlots]].
self computingAsp value = #foreground
ifTrue: [actionBlock value]
ifFalse: [UserProcessManager actualManager startBlock: actionBlock]
!OptimizationTool methodsFor: 'private'
actualMultiPlotAspects
| forAll optSymbol special |
| optSymbol := opt optTypeAsp value.
| forAll := #(fitTestAtAllVecPlotAsp #bestOfPopVecPlotAsp #worstOfPopVecPlotAsp ).
| special := #optSymbol = #ppbGeneral | | [optSymbol = #ppb1] [optSymbol = #chw1]
| [ifFalse: #fitTestAtAllDecVecPlotAsp ifTrue: #bestOfPopDecVecPlotAsp ]
| ifFalse: #fitTestAtAllDecVecPlotAsp]
| ifFalse: [#fitTestAtAllDecVecPlotAsp #bestOfPopDecVecPlotAsp #worstOfPopDecVecPlotAsp]
| ifFalse: [#fitTestAtAllDecVecPlotAsp]
| ifFalse: [#].

!forAll, special!
actualMultiPlotEncodedAspects
| forAllCode |
| forAllCode := self functionToBeEncoded
| ifTrue: [#fitTestAtAllDecVecPlotAsp #bestOfPopDecVecPlotAsp #worstOfPopDecVecPlotAsp]
| ifFalse: [#].

!forAllCode!
codeFunc: ABC
| from to | isCompiler evaluate: self fromVectorAsp value.

startData
**
| actionBlock |
| self indicesNecessary
ifTrue: [indices := Compiler evaluate: self indicesAsp value.
indices isNil ifTrue: ['Dialog warn: 'indices for plots necessary!!'].]
actionBlock :=
| self initDataDicts.
| self initOptimization.
| self startOptimization.
| self openPlotsAtEnd].
self computingAsp value = #foreground
ifTrue: [actionBlock value]
ifFalse: [UserProcessManager actualManager startBlock: actionBlock]
startOptimization
self optimization nextGenerations: self generationsAsp value.
Transcript cr; print: self optimization fitTestAtAll1; flush!
startPlot
**
| actionBlock |
| self indicesNecessary
ifTrue: [indices := Compiler evaluate: self indicesAsp value.
indices isNil ifTrue: ['Dialog warn: 'indices for plots necessary!!'].]
actionBlock :=
| self initDataDicts.
| self initOptimization.
| self computingAsp value = #begin ifTrue: [self openPlots].
| self startOptimization.
| self openPlotsAtEnd].
self computingAsp value = #foreground
ifTrue: [actionBlock value]
ifFalse: [UserProcessManager actualManager startBlock: actionBlock]
startOptimization
self optimization nextGenerations: self generationsAsp value.
Transcript cr; print: self optimization fitTestAtAll1; flush!
startPlot
**
| actionBlock |
| self indicesNecessary
ifTrue: [indices := Compiler evaluate: self indicesAsp value.
indices isNil ifTrue: ['Dialog warn: 'indices for plots necessary!!'].]
actionBlock :=
| self initDataDicts.
| self initOptimization.
| self computingAsp value = #begin ifTrue: [self openPlots].
| self startOptimization.
| self openPlotsAtEnd].
self computingAsp value = #foreground
ifTrue: [actionBlock value]
ifFalse: [UserProcessManager actualManager startBlock: actionBlock]
Smalltalk-Sourcen

Smalltalk-Sourcen

Aug 7 1997 23:06:11

Aug 7 1997 23:06:11

self bestOfPopQualAsp value: (optimization bestOfPopulation isNil ifTrue: [nil] ifFalse: [optimization bestOfPopulation quality]).
(optimization isKindOf: PBVectors)
ifTrue: [self vecSizeAsp value: optimization vectorSize].
(optimization isKindOf: PBILGeneral)
ifTrue: [self optTypeAsp value: #pBILGeneral].
self numOfBestAsp value: optimization numOfBest.
self mutProbabilityAsp value: optimization mutProbability.
self mutShiftAsp value: optimization mutShift.
self learningRateAsp value: optimization learningRate.
(optimization isKindOf: PBIL)
ifTrue: [self optTypeAsp value: #pBIL].
(optimization isKindOf: SBBVO)
ifTrue: [self optTypeAsp value: #sBBv].
self optimization: self optimization.
self moveMethodAsp value: optimization moveMethod.
setDecodeVecBC
| to from |
from := Compiler evaluate: self fromVectorAsp value.
to := Compiler evaluate: self toVectorAsp value.
(optimization isKindOf: PBIL)
ifTrue: [decodeVecBC := [:vec | vec asN undefinedFromBytesMin: maxVac: to]].

self functionTypeAsp value: #16BitsToNum ifTrue: [decodeVecBC := [:vec | vec asN undefinedFromBytesMin: maxVac: to]].

self functionTypeAsp value: #32BitsToNum ifTrue: [decodeVecBC := [:vec | vec asN undefinedFromBytesMin: maxVac: to]].
singlePlotAspects

** "#(bestOfPopQualPlotAsp worstOfPopQualPlotAsp fittestAtAllQualPlotAsp)" !

unknownFunctionErr
'self error: 'unknown aspect'!

unknownAspValErr
'self error: 'unknown aspect value'!

windowEvent: anEvent from: anApplicationWindow
'Release resources when the application is closed.'

super windowEvent: anEvent from: anApplicationWindow.

anEvent key == #close ifTrue: [self release] ! !

!OptimizationTool methodsFor: 'accessing'!

fittestAtAllVecValueHolder
"fittestAtAllVecValueHolder"

functionToBeCoded
"functionToBeCoded"

onOptimization: anOptimizationModel
self optValue: optimization optValue.
self readAspects.

optimization

self readAspects do: [:symbol | multiPlotDict at: symbol put: ((MultiPlotModel emptyPlots: in dices size) evaluationsAsp: #evaluationsAsp numOfBestAsp: #numOfBestAsp mutProbabilityAsp: #mutProbabilityAsp mutShiftAsp: #mutShiftAsp learningRateAsp: #learningRateAsp sigmarReducedFactorAsp: #sigmarReducedFactorAsp sigmarTargetAsp: #sigmarTargetAsp schwellE1StringAsp: #schwellE1StringAsp MonteCarloSizeAsp: #MonteCarloSizeAsp fromVectorAsp: #fromVectorAsp toVectorAsp: #toVectorAsp)].

self plotDists value := #no.
ifTrue: [self initPlotDict.

self initPlotMethods! :multiPlotDict.

operateOnBits: anOptSymbol.

anOptSymbol = #pBILGeneral ifTrue: [#pBIL]

anOptSymbol = #schwellE1 ifTrue: [self functionTypeAsp value: #schwellE1].

anOptSymbol = #learningRate ifTrue: [self functionTypeAsp value: #learningRate].

self functionTypeAsp value: #mutShift.

self functionTypeAsp value: #mutProbability.

self functionTypeAsp value: #numOfBest.

self functionTypeAsp value: #evaluations.

self functionTypeAsp value: #generations.

self functionTypeAsp value: #vectorSize.

self functionTypeAsp value: #randNo.

self functionTypeAsp value: #randSeed.

self functionTypeAsp value: #gamma.

self functionTypeAsp value: #beta.

self functionTypeAsp value: #alpha.

self functionTypeAsp value: #sigma.

self functionTypeAsp value: #weight.

self functionTypeAsp value: #seed.

vectors := self readAspects.

optimization isKindOf: Optimization.
ifTrue: [self functionAsp value: optimization quaBCString].
(optimization isKindOf: PopulationBasedOptimization)
ifTrue: [self randNoAsp value: optimization randNo].
self randSeedAsp value: optimization randSeed.
self evaluationAsp value: optimization evaluations.
self generationAsp value: optimization generations.
self fittestAtAllAsp value: optimization fittestAtAll isNil ifTrue: [nil] ifFalse: [optimization fittestAtAll quality]].
optimization notNil ifTrue: [optimizationModel addDependent: self]]]]

!OptimizationTool methodsFor: 'aspects'

bestOffPopDecVecPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

bestOffPopDecVecPlotAsp isNil ifTrue: [bestOffPopDecVecPlotAsp := false asValue] ifFalse: [bestOffPopDecVecPlotAsp]!

bestOffPopQualAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

bestOffPopQualAsp isNil ifTrue: [bestOffPopQualAsp := 0 asValue] ifFalse: [bestOffPopQualAsp]!

bestOffPopQualPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

bestOffPopQualPlotAsp isNil ifTrue: [bestOffPopQualPlotAsp := false asValue] ifFalse: [bestOffPopQualPlotAsp]!

bestOffPopVecPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

bestOffPopVecPlotAsp isNil ifTrue: [bestOffPopVecPlotAsp := false asValue] ifFalse: [bestOffPopVecPlotAsp]!

computingAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

computingAsp isNil ifTrue: [computingAsp := nil asValue] ifFalse: [computingAsp]!

evaluationAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

evaluationAsp isNil ifTrue: [evaluationAsp := 0 asValue] ifFalse: [evaluationAsp]!

evaluationAsp isNil ifTrue: [evaluationAsp := 0 asValue] ifFalse: [evaluationAsp]!

fittestAtAllDecVecPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

fittestAtAllDecVecPlotAsp isNil ifTrue: [fittestAtAllDecVecPlotAsp := false asValue] ifFalse: [fittestAtAllDecVecPlotAsp]!

fittestAtAllQualAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

fittestAtAllQualAsp isNil ifTrue: [fittestAtAllQualAsp := 0 asValue] ifFalse: [fittestAtAllQualAsp]!

fittestAtAllQualPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

fittestAtAllQualPlotAsp isNil ifTrue: [fittestAtAllQualPlotAsp := false asValue] ifFalse: [fittestAtAllQualPlotAsp]!

fittestAtAllVecPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

fittestAtAllVecPlotAsp isNil ifTrue: [fittestAtAllVecPlotAsp := false asValue] ifFalse: [fittestAtAllVecPlotAsp]!

fromVectorAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

fromVectorAsp isNil ifTrue: [fromVectorAsp := String new asValue] ifFalse: [fromVectorAsp]!

functionAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

functionAsp isNil ifTrue: [functionAsp := String new asValue] ifFalse: [functionAsp]!

functionTypeAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

functionTypeAsp isNil ifTrue: [functionTypeAsp := nil asValue] ifFalse: [functionTypeAsp]!

generationAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

generationAsp isNil ifTrue: [generationAsp := 0 asValue] ifFalse: [generationAsp]!

indicesAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

indicesAsp isNil ifTrue: [indicesAsp := String new asValue] ifFalse: [indicesAsp]!

learningRateAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

learningRateAsp isNil ifTrue: [learningRateAsp := 0 asValue] ifFalse: [learningRateAsp]!

minimizeAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

minimizeAsp isNil ifTrue: [minimizeAsp := nil asValue] ifFalse: [minimizeAsp]!

monteCarlosizeAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

monteCarlosizeAsp isNil ifTrue: [monteCarlosizeAsp := 0 asValue] ifFalse: [monteCarlosizeAsp]!

moveMethodAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"moveMethodAsp isNil ifTrue: [moveMethodAsp := nil asValue] ifFalse: [moveMethodAsp]!!

mueVecPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"mueVecPlotAsp isNil ifTrue: [mueVecPlotAsp := false asValue] ifFalse: [mueVecPlotAsp]!!

mutProbabilityAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"mutProbabilityAsp isNil ifTrue: [mutProbabilityAsp := 0 asValue] ifFalse: [mutProbabilityAsp]!!

mutShiftAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"mutShiftAsp isNil ifTrue: [mutShiftAsp := 0 asValue] ifFalse: [mutShiftAsp]!!

numOfBestAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"numOfBestAsp isNil ifTrue: [numOfBestAsp := 0 asValue] ifFalse: [numOfBestAsp]!!

optTypeAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"optTypeAsp isNil ifTrue: [optTypeAsp := nil asValue] ifFalse: [optTypeAsp]!!

plotsAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"plotsAsp isNil ifTrue: [plotsAsp := nil asValue] ifFalse: [plotsAsp]!!

popSizeAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"popSizeAsp isNil ifTrue: [popSizeAsp := 0 asValue] ifFalse: [popSizeAsp]!!

probabilityVecPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"probabilityVecPlotAsp isNil ifTrue: [probabilityVecPlotAsp := false asValue] ifFalse: [probabilityVecPlotAsp]!!

probabilityVectorPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"probabilityVectorPlotAsp isNil ifTrue: [probabilityVectorPlotAsp := false asValue] ifFalse: [probabilityVectorPlotAsp]!!

randNoAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"randNoAsp isNil ifTrue: [randNoAsp := 0 asValue] ifFalse: [randNoAsp]!!

randSeedAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"randSeedAsp isNil ifTrue: [randSeedAsp := 0 asValue] ifFalse: [randSeedAsp]!!

schwefelStringAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"schwefelStringAsp isNil ifTrue: [schwefelStringAsp := String new asValue] ifFalse: [schwefelStringAsp]!!

sigmaReduceFactorAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"sigmaReduceFactorAsp isNil ifTrue: [sigmaReduceFactorAsp := 0 asValue] ifFalse: [sigmaReduceFactorAsp]!!

sigmaTargetAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"sigmaTargetAsp isNil ifTrue: [sigmaTargetAsp := 0 asValue] ifFalse: [sigmaTargetAsp]!!

sigmaVecPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"sigmaVecPlotAsp isNil ifTrue: [sigmaVecPlotAsp := false asValue] ifFalse: [sigmaVecPlotAsp]!!

toVectorAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"toVectorAsp isNil ifTrue: [toVectorAsp := String new asValue] ifFalse: [toVectorAsp]!!

updateAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"updateAsp isNil ifTrue: [updateAsp := nil asValue] ifFalse: [updateAsp]!!

updateMethodsAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"updateMethodsAsp isNil ifTrue: [updateMethodsAsp := nil asValue] ifFalse: [updateMethodsAsp]!!

vecSizeAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"vecSizeAsp isNil ifTrue: [vecSizeAsp := 0 asValue] ifFalse: [vecSizeAsp]!!

worstOffPopDecVecPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"worstOffPopDecVecPlotAsp isNil ifTrue: [worstOffPopDecVecPlotAsp := false asValue] ifFalse: [worstOffPopDecVecPlotAsp]!!

worstOffPopQualPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"worstOffPopQualPlotAsp isNil ifTrue: [worstOffPopQualPlotAsp := false asValue] ifFalse: [worstOffPopQualPlotAsp]!!

worstOffPopVecPlotAsp
"This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method."

"worstOffPopVecPlotAsp isNil ifTrue: [worstOffPopVecPlotAsp := false asValue] ifFalse: [worstOffPopVecPlotAsp]!!