CODEA v3

1.0

Generated by Doxygen 1.7.2

Tue Jun 12 2012 16:33:27

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 moeoVRPUtils Namespace Reference

A set of structures and utility functions for the VRP-TW problem.

Classes

- struct ClientData

  Information regarding each client in the dataset. This structure is intended to be used to store the information of each client read from the data file.

Typedefs

- typedef struct moeoVRPUtils::ClientData ClientDataT

  Renaming of struct ClientData.

Functions

- void computeDistances ()

  Computes the distance between two clients. The computed distances will be stored in dist.

- void setTimeMatrixAsDistanceMatrix ()

  Set the time matrix to the distance matrix. This is needed in some datasets as they don't provide time information.

- void getTimeWindow (unsigned _client, double &_readyTime, double &_dueTime, double &_serviceTime)

Returns the time window information of a given client.

- double distance (unsigned _from, unsigned _to)

  A function to get the distance between two clients.

- double elapsedTime (unsigned _from, unsigned _to)

  A function to get the distance between two clients.

- float polarAngle (unsigned _from, unsigned _to)

  Computes de polar angle between clients.

- void loadDistanceMatrix (const char ∗_filename)

  Loads the problem distance matrix data from a given file.

- void loadTimeMatrix (const char ∗_filename)

  Loads the problem time matrix data from a given file.

- void load (const char ∗_fileName)

  Loads the problem data from a given file.

- void printRoute (const Route &_route)

  Prints a route to the standard output.

- void printRoutes (const Routes &_routes)

  Prints a set of routes to the standard output.

- void swap (Route &_routePlan, unsigned i, unsigned j)

  Swaps the position of two costumers.

- double travelDistance (const std::vector< int > &_routePlan)

  Evaluate the travel distance of a route plan.

- bool feasibleCapacity (const Route &_routePlan)

  Checks if a routePlan is feasible in terms of capacity.

- bool elapsedTimeBetweenTwoCostumers (double &_totalElapsedTime, unsigned &_numberOfViolations, unsigned _i, unsigned _j)

  Calculates the time that takes from a costumerto a costumer<j>.

- bool feasibleTimeWindows (const Route &_routePlan)

  Checks if a routePlan is feasible in terms of time windows.

- bool safetyCheck (const Route &_routePlan, std::string methodName="")
- bool safetyCheck (const std::vector< std::vector< int > > &_routePlan, std::string methodName="")
- void print ()

Variables

- long evaluations
- long numberOfOperations

5.1.1 Detailed Description

A set of structures and utility functions for the VRP-TW problem.

5.1.2 Typedef Documentation

5.1.2.1 typedef struct ClientData moeoVRPUtils::ClientDataT

Renaming of struct ClientData.

5.1.3 Function Documentation

5.1.3.1 void moeoVRPUtils::computeDistances ( )

Computes the distance between two clients. The computed distances will be stored in dist.

Here is the caller graph for this function:

5.1.3.2 double moeoVRPUtils::distance (unsigned *_from*, unsigned *_to*)

A function to get the distance between two clients.

Parameters

_from	The first client.
_to	The second client.

Returns

The distance between _from and _to.

Here is the caller graph for this function:

5.1.3.3 double moeoVRPUtils::elapsedTime (unsigned *_from*, unsigned *_to*)

A function to get the distance between two clients.

Parameters

_from	The first client.
_to	The second client.

Returns

The distance between _from and _to.

Here is the caller graph for this function:

5.1.3.4 bool moeoVRPUtils::elapsedTimeBetweenTwoCostumers (double & *_totalElapsedTime,* unsigned & *_numberOfViolations,* unsigned *_i,* unsigned *_j*)

Calculates the time that takes from a costumer*to a costumer<j>.*

Parameters

_- totalElapsedT	It's the travel time of a given route up to costumer.
_- numberOfViol	*Counts the number of violations or costumer not satisfaied within the given time.*
_i	*Index of the costumer the delivery vehicle departs from.*
_j	*Index of the costumer the delivery vehicle arrives at.*

Returns

<true> if is feasible (no costumer is served late) and <false> otherwise.

Here is the caller graph for this function:

5.1.3.5 bool moeoVRPUtils::feasibleCapacity (const Route & *_routePlan*)

Checks if a routePlan is feasible in terms of capacity.

Parameters

_routePlan	The route plan in which we want to swap costumers.

Returns

<true> if is feasible and <false> otherwise.

5.1.3.6 bool moeoVRPUtils::feasibleTimeWindows (const Route & *_routePlan*)

Checks if a routePlan is feasible in terms of time windows.

Parameters

_routePlan	The route plan to be checked.

Returns

<true> if is feasible and <false> otherwise.

Here is the call graph for this function:

5.1.3.7 void moeoVRPUtils::getTimeWindow (unsigned *_client,* double & *_readyTime,* double & *_dueTime,* double & *_serviceTime*)

Returns the time window information of a given client.

Parameters

_client	The client whose information we want to know.
_readyTime	Return value. The beginning of the client's time window.
_dueTime	Return value. The end of the client's time window.
_-serviceTime	Return value. The client's service time.

Here is the caller graph for this function:

5.1.3.8 void moeoVRPUtils::load (const char ∗ *_fileName*)

Loads the problem data from a given file.

Parameters

_fileName	The file to load data from.

Warning

No error check is performed!

Reading the header of the file

Here is the caller graph for this function:

5.1.3.9 void moeoVRPUtils::loadDistanceMatrix (const char ∗ *_filename*)

Loads the problem distance matrix data from a given file.

Parameters

_fileName	The file to load distance matrix data from.

Warning

No error check is performed!

Here is the caller graph for this function:

5.1.3.10 void moeoVRPUtils::loadTimeMatrix (const char ∗ _filename)

Loads the problem time matrix data from a given file.

Parameters

_fileName	The file to load time matrix data from.

Warning

No error check is performed!

Here is the caller graph for this function:

5.1.3.11 float moeoVRPUtils::polarAngle (unsigned _from, unsigned _to)

Computes de polar angle between clients.

Parameters

_from	The first client.
_to	The second client.

Returns

The polar angle between _from and _to.

5.1.3.12 void moeoVRPUtils::print ( )

5.1.3.13 void moeoVRPUtils::printRoute (const Route & _route)

Prints a route to the standard output.

Parameters

_route	The route to print.

5.1.3.14 void moeoVRPUtils::printRoutes (const Routes & _routes)

Prints a set of routes to the standard output.

Parameters

_routes	The set of routes to print.

Here is the caller graph for this function:

5.1.3.15 bool moeoVRPUtils::safetyCheck (const std::vector< std::vector< int > > & *_routePlan*, std::string *methodName* = "")

Here is the call graph for this function:

5.1.3.16 bool moeoVRPUtils::safetyCheck (const Route & *_routePlan*, std::string *methodName* = "")

Here is the caller graph for this function:

5.1.3.17 void moeoVRPUtils::setTimeMatrixAsDistanceMatrix ( )

Set the time matrix to the distance matrix. This is needed in some datasets as they don't provide time information.

Here is the caller graph for this function:

5.1.3.18 void moeoVRPUtils::swap (Route & *_routePlan*, unsigned *i*, unsigned *j*)

Swaps the position of two costumers.

Parameters

_routePlan	The routePlan in which we want to swap costumers.
_i	Position of the first costumer.
_j	Position of the second costumer.

Here is the caller graph for this function:

5.1.3.19 double moeoVRPUtils::travelDistance (const std::vector< int > & *_routePlan*)

Evaluate the travel distance of a route plan.

Parameters

_routePlan	The route plan to calculate its length

Returns

length of the given route plan.

5.1.4 Variable Documentation

5.1.4.1 long moeoVRPUtils::evaluations

5.1.4.2 long moeoVRPUtils::numberOfOperations

Chapter 6

Class Documentation

6.1 agent< EOT > Class Template Reference

Contains propperties and methods agents work with at a high level.

`#include <agent.h>`

Collaboration diagram for agent< EOT >:

Public Member Functions

- agent ()
- agent (const unsigned _id, const concurrentQueue< tMessage > _inbox, neighborhood< EOT > ∗_recipients, const vector< phase< EOT > ∗ > _phases)
- agent (const EOT &_agent)
- agent & operator= (const agent &_agent)
- ∼agent ()
- unsigned getId () const
- bool isActive () const
- neighborhood< EOT > ∗ getNeighborhood () const
- vector< phase< EOT > ∗ > getPhases () const
- phase< EOT > ∗ getPhase (unsigned _phaseIndex)
- concurrentQueue< tMessage > ∗ getPointerToInbox ()
- void queueMessage (const tMessage &_message)
- const concurrentQueue< tMessage > & getInbox () const
- unsigned numberOfPhases () const
- void setId (const unsigned _id)
- void setActive (const bool _active)
- void setNeighborhood (neighborhood< EOT > ∗_recipients)
- void setInBox (const concurrentQueue< tMessage > &_inbox)
- void setPhases (const vector< phase< EOT > ∗ > _phases)
- void addPhase (phase< EOT > ∗_phase)
- void clearMailBox ()
- void startPhase (const unsigned _phaseIndex, eoPop< EOT > &_population)

6.1.1 Detailed Description

template< class EOT> class agent< EOT >

Contains propperties and methods agents work with at a high level. This class doesn't have directions on how an agent behaves, but a basic scheme of all the components it uses to carry out its operations. These components are the attributes (explained below) that it needs to cooperate, perform solving operations, save temporary information, etcetera.

Author

Group of Intelligent Computing - Universidad de La Laguna - 2008

6.1.2 Constructor & Destructor Documentation

6.1.2.1 template< class EOT> agent< EOT >::agent ( ) [inline]

Default constructor. It does nothing.

6.1.2.2 template< class EOT> agent< EOT >::agent (const unsigned *_id,* const concurrentQueue< tMessage > *_inbox,* neighborhood< EOT > * *_recipients,* const vector< phase< EOT > * > *_phases*) [inline]

Standard constructor.

Parameters

idAgentType	is the identificator for the agent.
*core**	is a pointer to the core of the agent.
deque<mess	is the data structure the agent is going to use to store the incoming messages.
neighborhood	is the list of agents it is going to communicate with.
vector<phase	is the list of phases it is going to go through.

Returns

an agent with the given parameters.

6.1.2.3 template< class EOT> agent< EOT >::agent (const EOT & *_agent*) [inline]

6.1.2.4 template< class EOT> agent< EOT >::∼agent ( ) [inline]

Default destructor. It frees all the memory used by this class.

6.1.3 Member Function Documentation

6.1.3.1 template<class EOT> void agent< EOT >::addPhase (phase< EOT > ∗ *_phase*) [inline]

Method that adds a phase the agent's phases vector.

Parameters

agent's	phase.

6.1.3.2 template<class EOT> void agent< EOT >::clearMailBox ( ) [inline]

6.1.3.3 template<class EOT> unsigned agent< EOT >::getId ( ) const [inline]

Method that returns the agent's id.

Returns

agent's id.

6.1.3.4 template<class EOT> const concurrentQueue<tMessage>& agent< EOT >::getInbox ( ) const [inline]

Method that returns the agent's mailbox.

Returns

agent's mailbox.

6.1.3.5 template<class EOT> neighborhood<EOT>∗ agent< EOT >::getNeighborhood ( ) const [inline]

Method that returns the agent's neighborhood.

Returns

a pointer to agent's neighborhood.

Here is the caller graph for this function:

6.1.3.6 template<class EOT> phase<EOT>∗ agent< EOT >::getPhase (unsigned *_phaseIndex*) [inline]

6.1.3.7 template<class EOT> vector<phase<EOT>∗> agent< EOT >::getPhases ( ) const [inline]

Method that returns the agent's phases.

Returns

agent's phases.

6.1.3.8 template<class EOT> concurrentQueue<tMessage>* agent< EOT >::getPointerToInbox ( ) `[inline]`

Method that returns the agent's mailbox.

Returns

a pointer to agent's mailbox.

6.1.3.9 template<class EOT> bool agent< EOT >::isActive ( ) const `[inline]`

Method that returns whether the agent is active or not.

Returns

agent's state.

6.1.3.10 template<class EOT> unsigned agent< EOT >::numberOfPhases ( ) const `[inline]`

6.1.3.11 template<class EOT> agent& agent< EOT >::operator= (const agent< EOT > & *_agent*) `[inline]`

6.1.3.12 template<class EOT> void agent< EOT >::queueMessage (const tMessage & *_message*) `[inline]`

Method to queue a new message.

6.1.3.13 template<class EOT> void agent< EOT >::setActive (const bool *_active*) `[inline]`

Method that sets the agent's state.

Parameters

agent's	state.

6.1.3.14 template<class EOT> void agent< EOT >::setId (const unsigned *_id*) `[inline]`

Method that sets the agent's id.

Parameters

agent's	id.

6.1.3.15 template<class EOT> void agent< EOT >::setInBox (const concurrentQueue< tMessage > & *_inbox*) [inline]

Method that sets the agent's mailbox.

Parameters

agent's	mailbox.

6.1.3.16 template<class EOT> void agent< EOT >::setNeighborhood (neighborhood< EOT > * *_recipients*) [inline]

Method that sets the agent's neighborhood.

Parameters

a	pointer to agent's neighborhood.

6.1.3.17 template<class EOT> void agent< EOT >::setPhases (const vector< phase< EOT > * > *_phases*) [inline]

Method that sets the agent's phases.

Parameters

agent's	phases.

6.1.3.18 template<class EOT> void agent< EOT >::startPhase (const unsigned *_phaseIndex,* eoPop< EOT > & *_population*) [inline]

Method that invokes an agent's phase

Parameters

agent's	phase index

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

- core/agent.h

6.2 moeoVRPUtils::ClientData Struct Reference

Information regarding each client in the dataset. This structure is intended to be used to store the information of each client read from the data file.

```
#include <moeoVRPUtils.h>
```

Public Attributes

- unsigned id
- double x
- double y
- double demand
- double readyTime
- double dueTime
- double serviceTime

6.2.1 Detailed Description

Information regarding each client in the dataset. This structure is intended to be used to store the information of each client read from the data file.

6.2.2 Member Data Documentation

6.2.2.1 double moeoVRPUtils::ClientData::demand

Client's demand of delivered product.

6.2.2.2 double moeoVRPUtils::ClientData::dueTime

Client's end of the time window.

6.2.2.3 unsigned moeoVRPUtils::ClientData::id

Client ID number.

6.2.2.4 double moeoVRPUtils::ClientData::readyTime

Client's beginning of the time window.

6.2.2.5 double moeoVRPUtils::ClientData::serviceTime

Client's service time (time needed to serve the product).

6.2.2.6 double moeoVRPUtils::ClientData::x

Client's 'x' position in the map.

6.2.2.7 double moeoVRPUtils::ClientData::y

Client's 'y' position in the map.

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

- problems/VRPTW/moeoVRPUtils.h

6.3 concurrentQueue< Data > Class Template Reference

```
#include <concurrentQueue.h>
```

Public Member Functions

- concurrentQueue ()
- ∼concurrentQueue ()
- boost::mutex getMutex () const
- boost::condition_variable getConditionVariable () const
- void push (Data const &data)
- bool empty () const
- size_t size () const
- bool try_pop (Data &popped_value)
- void wait_and_pop (Data &popped_value)
- void clear ()
- std::queue< Data > getQueue () const
- void setQueue (const std::queue< Data > &newQueue)

template<typename Data> class concurrentQueue< Data >

6.3.1 Constructor & Destructor Documentation

6.3.1.1 template<typename Data > concurrentQueue< Data >::concurrentQueue ( )

Default constructor. It does nothing.

6.3.1.2 template<typename Data > concurrentQueue< Data >::∼concurrentQueue ( )

Destructor. It frees the_queue.

6.3.2 Member Function Documentation

6.3.2.1 template< typename Data > void concurrentQueue< Data >::clear ( )

Method that clear the queue up

Here is the caller graph for this function:

6.3.2.2 template< typename Data > bool concurrentQueue< Data >::empty ( ) const

Method that checks whether the queue is empty

Returns

bool checks the state

Here is the caller graph for this function:

6.3.2.3 template< typename Data > boost::condition_variable concurrentQueue< Data >::getConditionVariable ( ) const

Method that returns the condition variable.

Returns

a boost::condition_variable containing the condition variable

6.3.2.4 template< typename Data > boost::mutex concurrentQueue< Data >::getMutex ( ) const

Method that returns the mutex.

Returns

a boost::mutex containving the mutex

6.3.2.5 template< typename Data > std::queue< Data > concurrentQueue< Data >::getQueue ( ) const

Method that returns the internal object the queue.

Returns

std::queue is the object itself

Here is the caller graph for this function:

6.3.2.6 template<typename Data> void concurrentQueue< Data >::push (Data const & *data*)

Thread-safe method to push data into the queue

Parameters

Data	const& is a const reference to the data to be pushed

Here is the caller graph for this function:

6.3.2.7 template<typename Data> void concurrentQueue< Data >::setQueue (const std::queue< Data > & *newQueue*)

Method to establish the queue.

Parameters

const	std::queue<>& is a constant reference to the queue to be set up.

Here is the caller graph for this function:

6.3.2.8 template<typename Data > size_t concurrentQueue< Data >::size ( ) const

Method that returns the size of the queue

Returns

size_t is the number of elements the queue holds

6.3.2.9 template<typename Data> bool concurrentQueue< Data >::try_pop (Data & *popped_value*)

Method to pop data out of the queue

Parameters

Data&	is the reference to the data to be popped out

Returns

true or false if the operation was correctly done or not

6.3.2.10 template<typename Data> void concurrentQueue< Data >::wait_and_pop (Data & *popped_value*)

Thread-safe method to pop data out of the queue

Parameters

Data&	is the reference to the data to be popped out

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

- libs/concurrentQueue.h

6.4 container Class Reference

Allows to store a pair compound by an id and any type of information.

`#include <container.h>`

Collaboration diagram for container:

Public Member Functions

- container ()
- container (const string _id, const any _content)
- ∼container ()
- string getId () const
- any getContent () const
- any ∗ getPointerToContent ()
- void setId (const string _id)
- void setObject (const any _content)
- void set (const string _id, const any _content)

6.4.1 Detailed Description

Allows to store a pair compound by an id and any type of information. A message consists of a vector of this data structure (object).

Basic schema for a message :

```
________________________________________________________
||''''''''''''''''''''||''''''''''''''''''''|           |
||   id1   |  object1 ||   id2   |  object2 |    ...    |
||....................||....................|           |
|     container1             container2                 |
|_______________________________________________________|
                       message
```

Author

Group of Intelligent Computing - Universidad de La Laguna - 2008

6.4.2 Constructor & Destructor Documentation

6.4.2.1 container::container ( ) [inline]

Default constructor. It does nothing.

6.4.2.2 container::container (const string *_id,* const any *_content*) [inline]

Standard constructor.

Parameters

const	std::string is the id of the information to be stored.
const	boost::any is the object to be stored

6.4.2.3 container::∼container ( ) [inline]

Default destructor. It does nothing.

6.4.3 Member Function Documentation

6.4.3.1 any container::getContent ( ) const [inline]

Method that returns the object stored.

Returns

a boost object that wraps the information.

6.4.3.2 string container::getId ( ) const [inline]

Method that returns the identifier of the information stored.

Returns

a standard string that contains the id of the object stored.

6.4.3.3 any∗ container::getPointerToContent ( ) [inline]

Method that returns the object stored.

Returns

a pointer to a boost object that wraps the information.

6.4.3.4 void container::set (const string *_id,* const any *_content*) [inline]

Method that sets the both the id and the boost object to be saved.

Parameters

const	std::string is id of the information stored.
const	boost::any is object that will be stored.

Here is the caller graph for this function:

6.4.3.5 void container::setId (const string *_id*) [inline]

Method that sets the id of the information to be saved.

Parameters

const	std::string is id of the information stored.

6.4.3.6 void container::setObject (const any *_content*) [inline]

Method that sets the boost object to be saved.

Parameters

const	boost::any is object that will be stored.

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

- core/container.h

6.5 defaultCommunicationPhase< EOT > Class Template Reference

Carries out a by-default Communication Phase.

`#include <defaultCommunicationPhase.h>`

Inheritance diagram for defaultCommunicationPhase< EOT >:

Collaboration diagram for defaultCommunicationPhase< EOT >:

Public Member Functions

- defaultCommunicationPhase ()
- ∼defaultCommunicationPhase ()
- defaultCommunicationPhase< EOT > ∗ clone () const

- void createMessage (EOT ∗_agent, eoPop< EOT > &_pop)
- void operator() (EOT ∗_agent, eoPop< EOT > &_pop)
- void operator() (eoPop< EOT > &_pop)

6.5.1 Detailed Description

template<class EOT> class defaultCommunicationPhase< EOT >

Carries out a by-default Communication Phase. This class is designed to implement the steps to share information with other agents. To do this, it starts retrieving the list of recipients depending on the neighborhood of this agent. Next, it packs the message to be sent. And finally, it puts the message on neighbor's mailboxes.

Author

Group of Intelligent Computing - Universidad de La Laguna - 2008
ASAP Group - Nottingham University - 2011

6.5.2 Constructor & Destructor Documentation

6.5.2.1 template<class EOT > defaultCommunicationPhase< EOT >::defaultCommunicationPhase ( ) `[inline]`

Default constructor. It does nothing.

Here is the caller graph for this function:

6.5.2.2 template<class EOT > defaultCommunicationPhase< EOT >::∼defaultCommunicationPhase ( ) `[inline]`

Default destructor. It does nothing.

6.5.3 Member Function Documentation

6.5.3.1 template<class EOT > defaultCommunicationPhase<EOT>∗ defaultCommunicationPhase< EOT >::clone ( ) const `[inline, virtual]`

Implements phase< EOT >.

Here is the call graph for this function:

6.5.3.2 template<class EOT > void defaultCommunicationPhase< EOT >::createMessage (EOT ∗ *_agent,* eoPop< EOT > & *_pop*) `[inline]`

Here is the call graph for this function:

Here is the caller graph for this function:

6.5.3.3 template< class EOT > void defaultCommunicationPhase< EOT >::operator() (EOT ∗ , eoPop< EOT > &) [inline, virtual]

Method that is invoked from the ssystem's main-loop to start the phase. This simply calls the inherited methods explained above.

Implements phase< EOT >.

Here is the call graph for this function:

6.5.3.4 template< class EOT > void defaultCommunicationPhase< EOT >::operator() (eoPop< EOT > & *_pop*) [inline, virtual]

Implements phase< EOT >.

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

- agents/defaultCommunicationPhase.h

6.6 dummyPhase< EOT > Class Template Reference

```
#include <dummyPhase.h>
```

Inheritance diagram for dummyPhase< EOT >:

Collaboration diagram for dummyPhase< EOT >:

Public Member Functions

- void prePhase (EOT ∗_agent, eoPop< EOT > ∗_pop)
- void corePhase (EOT ∗_agent, eoPop< EOT > ∗_pop)
- void postPhase (EOT ∗_agent, eoPop< EOT > ∗_pop)
- void operator() (eoPop< EOT > &_pop)
- void operator() (EOT ∗agent, eoPop< EOT > &pop)

template<class EOT> class dummyPhase< EOT >

6.6.1 Member Function Documentation

6.6.1.1 template< class EOT > void dummyPhase< EOT >::corePhase (EOT ∗ *_agent,* eoPop< EOT > ∗ *_pop*) [inline]

6.6.1.2 template< class EOT > void dummyPhase< EOT >::operator() (EOT ∗ , eoPop< EOT > &) [inline, virtual]

Method that is invoked from the ssystem's main-loop to start the phase. This simply calls the inherited methods explained above.

Implements phase< EOT >.

6.6.1.3 template<class EOT > void dummyPhase< EOT >::operator() (eoPop< EOT > & *_pop*) `[inline, virtual]`

Implements phase< EOT >.

6.6.1.4 template<class EOT > void dummyPhase< EOT >::postPhase (EOT ∗ *_agent*, eoPop< EOT > ∗ *_pop*) `[inline]`

6.6.1.5 template<class EOT > void dummyPhase< EOT >::prePhase (EOT ∗ *_agent*, eoPop< EOT > ∗ *_pop*) `[inline]`

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

- dummyPhase.h

6.7 eoMonSingleGenOp< EOT > Class Template Reference

`#include <eoMonSingleGenOp.h>`

Public Member Functions

- eoMonSingleGenOp (eoMonOp< EOT > &_op)
- unsigned max_production (void)
- eoMonOp< EOT > & getOperators ()
- void apply (eoPopulator< EOT > &_it)
- bool apply (EOT &_indi)
- virtual std::string className () const

6.7.1 Detailed Description

template<class EOT> class eoMonSingleGenOp< EOT >

CODISEO Wrapper for MonOp

6.7.2 Constructor & Destructor Documentation

6.7.2.1 template<class EOT> eoMonSingleGenOp< EOT >::eoMonSingleGenOp (eoMonOp< EOT > & *_op*) `[inline]`

6.7.3 Member Function Documentation

6.7.3.1 template<class EOT> void eoMonSingleGenOp< EOT >::apply (eoPopulator< EOT > & *_it*) `[inline]`

Here is the caller graph for this function:

6.7.3.2 template<class EOT> bool eoMonSingleGenOp< EOT >::apply (EOT & *_indi*) `[inline]`

6.7.3.3 template<class EOT> virtual std::string eoMonSingleGenOp< EOT >::className ( ) const `[inline, virtual]`

6.7.3.4 template<class EOT> eoMonOp<EOT>& eoMonSingleGenOp< EOT >::getOperators ( ) `[inline]`

6.7.3.5 template<class EOT> unsigned eoMonSingleGenOp< EOT >::max_production (void) `[inline]`

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

- core/eoMonSingleGenOp.h

6.8 eoQuadSingleGenOp< EOT > Class Template Reference

```
#include <eoQuadSingleGenOp.h>
```

Public Member Functions

- eoQuadSingleGenOp (eoQuadOp< EOT > &_op)
- unsigned max_production (void)
- eoQuadOp< EOT > & getOperators ()
- void apply (eoPopulator< EOT > &_pop)
- bool apply (EOT &_indi1, EOT &_indi2)
- virtual std::string className () const

6.8.1 Detailed Description

template<class EOT> class eoQuadSingleGenOp< EOT >

CODISEO Wrapper for quadop

6.8.2 Constructor & Destructor Documentation

6.8.2.1 template<class EOT> eoQuadSingleGenOp< EOT >::eoQuadSingleGenOp (eoQuadOp< EOT > & *_op*) `[inline]`

6.8.3 Member Function Documentation

6.8.3.1 template<class EOT> void eoQuadSingleGenOp< EOT >::apply (eoPopulator< EOT > & *_pop*) `[inline]`

Here is the caller graph for this function:

6.8.3.2 template<class EOT> bool eoQuadSingleGenOp< EOT >::apply (EOT & *_indi1*, EOT & *_indi2*) `[inline]`

6.8.3.3 template<class EOT> virtual std::string eoQuadSingleGenOp< EOT >::className ( ) const `[inline, virtual]`

6.8.3.4 template<class EOT> eoQuadOp<EOT>& eoQuadSingleGenOp< EOT >::getOperators ( ) `[inline]`

6.8.3.5 template<class EOT> unsigned eoQuadSingleGenOp< EOT >::max_production (void) `[inline]`

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

- core/eoQuadSingleGenOp.h

6.9 eoSingleOp< EOT > Class Template Reference

`#include <eoSingleOp.h>`

Public Types

- typedef unsigned position_type

Public Member Functions

- eoSingleOp ()
- void apply (eoPopulator< EOT > &_pop)
- bool apply (EOT &_individual1, EOT &_individual2)
- bool apply (EOT &_individual)
- virtual std::string className () const

Protected Member Functions

- void findIndices ()

6.9.1 Detailed Description

template<class EOT> class eoSingleOp< EOT >

Single Operation: Only one operator is randomly applied. The operator is selected using the given rates.

6.9.2 Member Typedef Documentation

6.9.2.1 template<class EOT> typedef unsigned eoSingleOp< EOT >::position_type

6.9.3 Constructor & Destructor Documentation

6.9.3.1 template<class EOT> eoSingleOp< EOT >::eoSingleOp ( ) [inline]

6.9.4 Member Function Documentation

6.9.4.1 template<class EOT> void eoSingleOp< EOT >::apply (eoPopulator< EOT > & *_pop*) [inline]

Here is the caller graph for this function:

6.9.4.2 template<class EOT> bool eoSingleOp< EOT >::apply (EOT & *_individual*) [inline]

Here is the call graph for this function:

6.9.4.3 template<class EOT> bool eoSingleOp< EOT >::apply (EOT & *_individual1,* EOT & *_individual2*) [inline]

Here is the call graph for this function:

6.9.4.4 template< class EOT > virtual std::string eoSingleOp< EOT >::className ( ) const `[inline, virtual]`

Here is the caller graph for this function:

6.9.4.5 template< class EOT > void eoSingleOp< EOT >::findIndices ( ) `[inline, protected]`

Here is the call graph for this function:

Here is the caller graph for this function:

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

- core/eoSingleOp.h

6.10 eoVRPEvalFunc Class Reference

Evaluates an individual of type eoVRP.

6.10.1 Detailed Description

Evaluates an individual of type eoVRP.

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

- problems/VRPTW/moeoVRPEvalFunc.h

6.11 mailBox Class Reference

`#include <mailBox.h>`

Collaboration diagram for mailBox:

Public Member Functions

- mailBox ()
- ∼mailBox ()
- const concurrentQueue< tMessage > & getInbox () const
- concurrentQueue< tMessage > ∗ getPointerToInbox ()
- void setInbox (const concurrentQueue< tMessage > &_inbox)
- void clear ()
- void insert (const tMessage _message)
- void push_back (const tMessage &_message)
- bool empty () const

6.11.1 Constructor & Destructor Documentation

6.11.1.1 mailBox::mailBox ( ) `[inline]`

Default constructor. It does nothing.

6.11.1.2 mailBox::∼mailBox ( ) `[inline]`

Default destructor. It does nothing.

6.11.2 Member Function Documentation

6.11.2.1 void mailBox::clear ( ) `[inline]`

Method that clears up the mailTypebox.

Here is the call graph for this function:

Here is the caller graph for this function:

6.11.2.2 bool mailBox::empty ( ) const `[inline]`

Method that returns true or false depending on whether the mailTypebox is empty or not.

Parameters

bool	is true if the mailTypebox is empty, false otherwise.

Here is the call graph for this function:

6.11.2.3 const concurrentQueue<tMessage>& mailBox::getInbox ( ) const `[inline]`

Method that returns the mailTypebox.

Returns

the data structure that holds the messages.

Here is the caller graph for this function:

6.11.2.4 concurrentQueue<tMessage>∗ mailBox::getPointerToInbox ( ) `[inline]`

Method that returns the mailTypebox.

Returns

a pointer to the data structure that holds the messages.

Here is the caller graph for this function:

6.11.2.5 void mailBox::insert (const tMessage _*message*) [inline]

Method that inserts a new message in the mailTypebox.

Parameters

const	message is the new message to be inserted in the mailTypebox.

Here is the call graph for this function:

6.11.2.6 void mailBox::push_back (const tMessage & _*message*) [inline]

Method that inserts a new message in the mailTypebox.

Parameters

const	message is the new message to be inserted in the mailTypebox.

Note

This method was implemented to maintain compatibility with the STL.

Here is the call graph for this function:

Here is the caller graph for this function:

6.11.2.7 void mailBox::setInbox (const concurrentQueue< tMessage > & _*inbox*) [inline]

Method that sets the mailTypebox.

Parameters

const	mailType is the mailTypebox.

Here is the call graph for this function:

Here is the caller graph for this function:

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

- core/mailBox.h

6.12 mailTypeBox Class Reference

Stores the messages sent by other agents.

```
#include <mailBox.h>
```

6.12.1 Detailed Description

Stores the messages sent by other agents. This class handles basic methods to manage a virtual mailTypebox of message sent by other agents.

Author

Group of Intelligent Computing - Universidad de La Laguna - 2008

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

- core/mailBox.h

6.13 moeoCoverageMetric< ObjectiveVector > Class Template Reference

```
#include <moeoCoverageMetric.h>
```

Public Member Functions

- double operator() (const std::vector< ObjectiveVector > &_setA, const std::vector< ObjectiveVector > &_setB)

6.13.1 Detailed Description

template<class ObjectiveVector> class moeoCoverageMetric< ObjectiveVector >

The Coverage Metric measures the extent to which one solution set B is covered by another solution set A.

6.13.2 Member Function Documentation

6.13.2.1 template<class ObjectiveVector> double moeoCoverageMetric< ObjectiveVector >::operator() (const std::vector< ObjectiveVector > & *_setA*, const std::vector< ObjectiveVector > & *_setB*) [inline]

Returns the coverage of the Pareto set '_setB' relatively to the Pareto set '_setA'

Parameters

_setA	the first Pareto set
_setB	the second Pareto set

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

- core/moeoCoverageMetric.h

6.14 moeoJFOPhaseAlgorithm< EOT > Class Template Reference

```
#include <moeoJFOPhaseAlgorithm.h>
```

Inheritance diagram for moeoJFOPhaseAlgorithm< EOT >:

Collaboration diagram for moeoJFOPhaseAlgorithm< EOT >:

Public Member Functions

- moeoJFOPhaseAlgorithm (eoEvalFunc< EOT > &_eval, eoGenOp< EOT > &_operators, moMOLS< EOT, tObjectiveVector > &_localSearch, moeoParetoObjectiveVectorComparator< tObjectiveVector > &_comparator)
- moeoJFOPhaseAlgorithm (double _c1, double _c2, double _c3, double _c4, eoEvalFunc< EOT > &_eval, eoGenOp< EOT > &_operators)
- moeoJFOPhaseAlgorithm (const moeoJFOPhaseAlgorithm< EOT > &_phase)
- ∼moeoJFOPhaseAlgorithm ()
- void setCoefficients (double _c1, double _c2, double _c3, double _c4)
- void setCoefficients (const std::vector< double > &_coefficients)
- moeoJFOPhaseAlgorithm< EOT > ∗ clone () const
- void getBestNeighboringParticle (EOT ∗_particle)
- void operator() (EOT ∗_agent, eoPop< EOT > &_pop)
- void operator() (eoPop< EOT > &pop)

Static Public Member Functions

- static moeoUnboundedArchive< EOT > getArchive ()

Protected Member Functions

- void initialize (EOT ∗_particle, eoPop< EOT > &_pop)

template<class EOT> class moeoJFOPhaseAlgorithm< EOT >

6.14.1 Constructor & Destructor Documentation

6.14.1.1 template<class EOT> moeoJFOPhaseAlgorithm< EOT >::moeoJFOPhaseAlgorithm (eoEvalFunc< EOT > & *_eval,* eoGenOp< EOT > & *_operators,* moMOLS< EOT, tObjectiveVector > & *_localSearch,* moeoParetoObjectiveVectorComparator< tObjectiveVector > & *_comparator*) [inline]

Here is the caller graph for this function:

6.14.1.2 template<class EOT> moeoJFOPhaseAlgorithm< EOT >::moeoJFOPhaseAlgorithm (double *_c1,* double *_c2,* double *_c3,* double *_c4,* eoEvalFunc< EOT > & *_eval,* eoGenOp< EOT > & *_operators*) [inline]

6.14.1.3 template<class EOT> moeoJFOPhaseAlgorithm< EOT >::moeoJFOPhaseAlgorithm (const moeoJFOPhaseAlgorithm< EOT > & *_phase*) [inline]

6.14.1.4 template<class EOT> moeoJFOPhaseAlgorithm< EOT >::~moeoJFOPhaseAlgorithm ( ) [inline]

6.14.2 Member Function Documentation

6.14.2.1 template<class EOT> moeoJFOPhaseAlgorithm<EOT>∗ moeoJFOPhaseAlgorithm< EOT >::clone ( ) const [inline, virtual]

Implements phase< EOT >.

Here is the call graph for this function:

6.14.2.2 template<class EOT> static moeoUnboundedArchive<EOT> moeoJFOPhaseAlgorithm< EOT >::getArchive ( ) [inline, static]

6.14.2.3 template<class EOT> void moeoJFOPhaseAlgorithm< EOT >::getBestNeighboringParticle (EOT ∗ *_particle*) [inline]

Here is the caller graph for this function:

6.14.2.4 template<class EOT> void moeoJFOPhaseAlgorithm< EOT >::initialize (EOT ∗ *_particle,* eoPop< EOT > & *_pop*) [inline, protected]

Here is the caller graph for this function:

6.14.2.5 template<class EOT> void moeoJFOPhaseAlgorithm< EOT >::operator() (eoPop< EOT > & *pop*) [inline, virtual]

Implements phase< EOT >.

6.14.2.6 template<class EOT> void moeoJFOPhaseAlgorithm< EOT >::operator() (EOT ∗ , eoPop< EOT > &) [inline, virtual]

Method that is invoked from the ssystem's main-loop to start the phase. This simply calls the inherited methods explained above.

Implements phase< EOT >.

Here is the call graph for this function:

6.14.2.7 template<class EOT> void moeoJFOPhaseAlgorithm< EOT >::setCoefficients (double *_c1*, double *_c2*, double *_c3*, double *_c4*) [inline]

6.14.2.8 template<class EOT> void moeoJFOPhaseAlgorithm< EOT >::setCoefficients (const std::vector< double > & *_coefficients*) [inline]

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

- agents/moeoJFOPhaseAlgorithm.h

6.15 moeoStrictObjectiveVectorComparator< ObjectiveVector > Class Template Reference

`#include <moeoStrictObjectiveVectorComparator.h>`

Public Member Functions

- const bool operator() (const ObjectiveVector &_objectiveVector1, const ObjectiveVector &_objectiveVector2)

6.15.1 Detailed Description

template<class ObjectiveVector> class moeoStrictObjectiveVectorComparator< ObjectiveVector >

This functor class allows to compare 2 objective vectors according to strict dominance.

6.15.2 Member Function Documentation

6.15.2.1 template<class ObjectiveVector > const bool moeoStrictObjectiveVectorComparator< ObjectiveVector >::operator() (const ObjectiveVector & *_objectiveVector1*, const ObjectiveVector & *_objectiveVector2*) [inline]

Returns true if _objectiveVector1 is strictly dominated by _objectiveVector2

Parameters

_- *objectiveVecto*	the first objective vector
_- *objectiveVecto*	the second objective vector

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

- moeoStrictObjectiveVectorComparator.h

6.16 moeoVRP Class Reference

Defines the getoype used to solve the VRP-TW problem. Objectives:

- <Size of="" the="" fleet>="">int is number of vehicles we need to use to satisfy all costumers. – <Travel distance>="">double is length of the route-plan. – <Travel time>="">double is elapsed time since the first delivery vehicle departs from the depot until the last arrived at the depot. – <Waiting time>="">double is the amount of time that vehicles have to wait at each costumer location. – <Delay time>="">double is the amount of time by which the arrival of the vehicles + service time is retarded respect to the closing time of the costumers.

```
#include <moeoVRP.h>
```

Inheritance diagram for moeoVRP:

Collaboration diagram for moeoVRP:

Public Member Functions

- moeoVRP ()

  Default constructor: initializes variables to safe values.

- moeoVRP (const moeoVRP &_orig)

  Copy contructor: creates a new individual from a given one.

- ∼moeoVRP ()

  Default destructor: nothing to do here.

- moeoVRP & operator= (const moeoVRP &_orig)

  Performs a copy from the invidual passed as argument.

- virtual std::string className () const

  Returns a string containing the name of the class.

- void printOn (std::ostream &_os) const

  Prints the individual to a given stream.

- void printAllOn (std::ostream &_os) const

  Prints a detailed version of the individual (decoding information, unsatisfied contraints, etc.) to a given stream.

- void writeRoutePlan (std::string _filename) const

  Write a route-plan in a file.

- void readFrom (std::istream &_is)

  Reads an individual from a given stream.

- const Routes & routes () const

  Returns a reference to the decoded individual.

- void printRoutes (std::ostream &_os) const

  Aux. method to print a structure of routes.

- void printRoute (std::ostream &_os, unsigned _p) const

  Aux. method to print only one route.

- bool clean ()

  Cleans the individual (the vector of clients and also the decoding information).

- bool cleanRoutes ()

  Invalidates the decoding information (usually after crossover or mutation).

- bool decoded () const

  Has this individual been decoded?

- bool encode (Routes &_routes)

  Encodes an individual from a set of routes (usually used within crossover). The chromosome will have the following structure:

- bool decode ()

  Decodes an individual for its evaluation.

- unsigned sizeOfFleet ()

  Returns the number of vehicles need to satisfy all costumers in this route.

- void sizeOfFleet (unsigned mSizeOfFleet)

  Sets the size of the fleet.

- double length ()

  Returns the total cost (length) of traveling all the routes.

- void length (double mLength)

  Sets the lenght of the current route-plan.

- double time ()

  Returns the total cost (travel time) of traveling all routes.

- void time (double mTime)

  Sets the elapsed time of all routes.

- double waitingTime ()

  Returns the total amount of time vehicles have to wait for costumer to open their time windows.

- void waitingTime (double mWaitingTime)

  Sets the waiting time at all costumers.

- double delayTime ()

  Returns the total amount of exceed time serving costumers.

- void delayTime (double mDelayTime)

  Sets the total delay time.

6.16.1 Detailed Description

Defines the getoype used to solve the VRP-TW problem. Objectives:

- <Size of="" the="" fleet>="">int is number of vehicles we need to use to satisfy all costumers. – <Travel distance>="">double is length of the route-plan. – <Travel time>="">double is elapsed time since the first delivery vehicle departs from the depot until the last arrived at the depot. – <Waiting time>="">double is the amount of time that vehicles have to wait at each costumer location. – <Delay time>="">double is the amount of time by which the arrival of the vehicles + service time is retarded respect to the closing time of the costumers.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 moeoVRP::moeoVRP ( ) [inline]

Default constructor: initializes variables to safe values.

6.16.2.2 moeoVRP::moeoVRP (const moeoVRP & *_orig*) [inline]

Copy contructor: creates a new individual from a given one.

Parameters

_orig	The individual used to create the new one.

6.16.2.3 moeoVRP::∼moeoVRP ( ) [inline]

Default destructor: nothing to do here.

6.16.3 Member Function Documentation

6.16.3.1 virtual std::string moeoVRP::className ( ) const `[inline, virtual]`

Returns a string containing the name of the class.

Returns

The string containing the name of the class.

6.16.3.2 bool moeoVRP::clean ( ) `[inline]`

Cleans the individual (the vector of clients and also the decoding information).

Returns

True if the operation finishes correctly. False otherwise.

Here is the caller graph for this function:

6.16.3.3 bool moeoVRP::cleanRoutes ( ) `[inline]`

Invalidates the decoding information (usually after crossover or mutation).

Returns

True if the operation finishes correctly. False otherwise.

Here is the caller graph for this function:

6.16.3.4 bool moeoVRP::decode ( ) `[inline]`

Decodes an individual for its evaluation.

Returns

True if the operation finishes correctly. False otherwise.

Here is the call graph for this function:

Here is the caller graph for this function:

6.16.3.5 bool moeoVRP::decoded ( ) const `[inline]`

Has this individual been decoded?

Returns

True if has decoding information. False otherwise.

Here is the caller graph for this function:

6.16.3.6 double moeoVRP::delayTime ( ) [inline]

Returns the total amount of exceed time serving costumers.

Returns

The total delay time in all routes.

Here is the caller graph for this function:

6.16.3.7 void moeoVRP::delayTime (double *mDelayTime*) [inline]

Sets the total delay time.

Parameters

mDelayTime	is the value of the total delay.

6.16.3.8 bool moeoVRP::encode (Routes & *_routes*) [inline]

Encodes an individual from a set of routes (usually used within crossover). The chromosome will have the following structure:

[Route_1] 0 [Route_2] 0 [Route_3] 0 ... 0 [Route_n]

Returns

True if the operation finishes correctly. False otherwise.

Here is the call graph for this function:

Here is the caller graph for this function:

6.16.3.9 double moeoVRP::length ( ) [inline]

Returns the total cost (length) of traveling all the routes.

Returns

The total cost (length) of traveling all the routes.

Here is the caller graph for this function:

6.16.3.10 void moeoVRP::length (double *mLength*) [inline]

Sets the lenght of the current route-plan.

Parameters

mLength	is the value of the length to be set up.

6.16.3.11 moeoVRP& moeoVRP::operator= (const moeoVRP & *_orig*) [inline]

Performs a copy from the invidual passed as argument.

Parameters

_orig	The individual to copy from.

Returns

A reference to this.

Here is the call graph for this function:

6.16.3.12 void moeoVRP::printAllOn (std::ostream & *_os*) const [inline]

Prints a detailed version of the individual (decoding information, unsatisfied contraints, etc.) to a given stream.

Parameters

_os	The stream to print to.

Here is the call graph for this function:

6.16.3.13 void moeoVRP::printOn (std::ostream & *_os*) const [inline]

Prints the individual to a given stream.

Parameters

_os	The stream to print to.

Here is the call graph for this function:

Here is the caller graph for this function:

6.16.3.14 void moeoVRP::printRoute (std::ostream & *_os,* unsigned *_p*) const [inline]

Aux. method to print only one route.

Parameters

_os	The stream to print to.
_p	The route to print.

6.16.3.15 void moeoVRP::printRoutes (std::ostream & *_os*) const [inline]

Aux. method to print a structure of routes.

Parameters

_os	The stream to print to.

Here is the caller graph for this function:

6.16.3.16 void moeoVRP::readFrom (std::istream & *_is*) [inline]

Reads an individual from a given stream.

Parameters

_is	The stream to read from.

6.16.3.17 const Routes& moeoVRP::routes ( ) const [inline]

Returns a reference to the decoded individual.

Returns

A reference to the decoded individual.

Here is the caller graph for this function:

6.16.3.18 unsigned moeoVRP::sizeOfFleet ( ) [inline]

Returns the number of vehicles need to satisfy all costumers in this route.

Returns

The total number of vehicles used in this route-plan.

Here is the caller graph for this function:

6.16.3.19 void moeoVRP::sizeOfFleet (unsigned *mSizeOfFleet*) [inline]

Sets the size of the fleet.

Parameters

mSizeOfFleet	is the size of the fleet to be set up.

6.16.3.20 void moeoVRP::time (double *mTime*) [inline]

Sets the elapsed time of all routes.

Parameters

mTime	is the elapsed time to be set up.

6.16.3.21 double moeoVRP::time ( ) [inline]

Returns the total cost (travel time) of traveling all routes.

Returns

The total cost (travel time) of traveling all routes.

Here is the caller graph for this function:

6.16.3.22 double moeoVRP::waitingTime ( ) [inline]

Returns the total amount of time vehicles have to wait for costumer to open their time windows.

Returns

The total amount of waiting time at all costumers.

Here is the caller graph for this function:

6.16.3.23 void moeoVRP::waitingTime (double *mWaitingTime*) [inline]

Sets the waiting time at all costumers.

Parameters

mWaitingTime	is the total ammount of waiting time

6.16.3.24 void moeoVRP::writeRoutePlan (std::string *_filename*) const [inline]

Write a route-plan in a file.

Parameters

_filename	is the name of the file the route-plan will be written in

Here is the call graph for this function:

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

- problems/VRPTW/moeoVRP.h

6.17 moeoVRPDisplacementMutation Class Reference

Implementation of the displacement mutation operator.

`#include <moeoVRPMutation.h>`

Public Member Functions

- moeoVRPDisplacementMutation ()

  Deafult constructor.

- std::string className () const

  Returns a string containing the name of the class. Used to display statistics.

- bool operator() (moeoVRP &_genotype)

  It selects a set of clients, erases them from their original position and inserts them somewhere else. The selected set of clients may cover different routes.

6.17.1 Detailed Description

Implementation of the displacement mutation operator.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 moeoVRPDisplacementMutation::moeoVRPDisplacementMutation ( ) `[inline]`

Deafult constructor.

6.17.3 Member Function Documentation

6.17.3.1 std::string moeoVRPDisplacementMutation::className ( ) const `[inline]`

Returns a string containing the name of the class. Used to display statistics.

Returns

The string containing the name of the class.

6.17.3.2 bool moeoVRPDisplacementMutation::operator() (moeoVRP & *_genotype*) [inline]

It selects a set of clients, erases them from their original position and inserts them somewhere else. The selected set of clients may cover different routes.

Parameters

_genotype	The genotype being mutated (it will be probably modified).

Returns

True if the individual has been modified. False otherwise.

Here is the call graph for this function:

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

- problems/VRPTW/moeoVRPMutation.h

6.18 moeoVRPEdgeCrossover Class Reference

Implementation of the classic Edge Crossover from the TSP.

`#include <moeoVRPQuadCrossover.h>`

Public Member Functions

- moeoVRPEdgeCrossover ()

  Deafult constructor.

- std::string className () const

  Returns a string containing the name of the class. Used to display statistics.

- bool operator() (moeoVRP &_genotype1, moeoVRP &_genotype2)

  Both parameters are the parents and the (future) children of the crossover.

6.18.1 Detailed Description

Implementation of the classic Edge Crossover from the TSP.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 moeoVRPEdgeCrossover::moeoVRPEdgeCrossover ( ) [inline]

Deafult constructor.

6.18.3 Member Function Documentation

6.18.3.1 std::string moeoVRPEdgeCrossover::className ( ) const [inline]

Returns a string containing the name of the class. Used to display statistics.

Returns

The string containing the name of the class.

6.18.3.2 bool moeoVRPEdgeCrossover::operator() (moeoVRP & *_genotype1,* moeoVRP & *_genotype2*) [inline]

Both parameters are the parents and the (future) children of the crossover.

Parameters

_genotype1	The first parent.
_genotype2	The second parent.

Returns

True if any of the parents was modified. False otherwise.

Here is the call graph for this function:

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

- problems/VRPTW/moeoVRPQuadCrossover.h

6.19 moeoVRPEvalFunc Class Reference

```
#include <moeoVRPEvalFunc.h>
```

Public Member Functions

- moeoVRPEvalFunc ()

  Constructor: nothing to do here.
- ∼moeoVRPEvalFunc ()

  Destructor: nothing to do here.
- void operator() (moeoVRP &_moeo)

  Invokes all evaluators.
- unsigned sizeOfFleet (const moeoVRP &_moeo) const

Computes the number of routes of the individual.

- double travelDistance (const moeoVRP &_moeo) const

  Computes the travel distance of the individual.

- double travelTime (const moeoVRP &_moeo) const

  Computes the travel time of the individual.

- double waitingTime (const moeoVRP &_moeo) const

  Computes the total waiting time of the individual.

- double delayTime (const moeoVRP &_moeo) const

  Computes the total delay of the individual.

6.19.1 Constructor & Destructor Documentation

6.19.1.1 moeoVRPEvalFunc::moeoVRPEvalFunc ( ) [inline]

Constructor: nothing to do here.

6.19.1.2 moeoVRPEvalFunc::∼moeoVRPEvalFunc ( ) [inline]

Destructor: nothing to do here.

6.19.2 Member Function Documentation

6.19.2.1 double moeoVRPEvalFunc::delayTime (const moeoVRP & *_moeo*) const [inline]

Computes the total delay of the individual.

Parameters

_moeo	The individual to be evaluated.

Here is the call graph for this function:

Here is the caller graph for this function:

6.19.2.2 void moeoVRPEvalFunc::operator() (moeoVRP & *_moeo*) [inline]

Invokes all evaluators.

Parameters

_moeo	The individual to be evaluated.

Here is the call graph for this function:

6.19.2.3 unsigned moeoVRPEvalFunc::sizeOfFleet (const moeoVRP & *_moeo*) const `[inline]`

Computes the number of routes of the individual.

Parameters

_moeo	The individual to be evaluated.

Here is the call graph for this function:

Here is the caller graph for this function:

6.19.2.4 double moeoVRPEvalFunc::travelDistance (const moeoVRP & *_moeo*) const `[inline]`

Computes the travel distance of the individual.

Parameters

_moeo	The individual to be evaluated.

Here is the call graph for this function:

Here is the caller graph for this function:

6.19.2.5 double moeoVRPEvalFunc::travelTime (const moeoVRP & *_moeo*) const `[inline]`

Computes the travel time of the individual.

Parameters

_moeo	The individual to be evaluated.

Here is the call graph for this function:

Here is the caller graph for this function:

6.19.2.6 double moeoVRPEvalFunc::waitingTime (const moeoVRP & *_moeo*) const `[inline]`

Computes the total waiting time of the individual.

Parameters

_moeo	The individual to be evaluated.

Here is the call graph for this function:

Here is the caller graph for this function:

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

- problems/VRPTW/moeoVRPEvalFunc.h

6.20 moeoVRPGenericCrossover Class Reference

Implementation of the generic crossover for the VRP-TW by Tavares et al.

`#include <moeoVRPQuadCrossover.h>`

Public Member Functions

- moeoVRPGenericCrossover ()

  Deafult constructor.

- std::string className () const

  Returns a string containing the name of the class. Used to display statistics.

- bool operator() (moeoVRP &_genotype1, moeoVRP &_genotype2)

  Both parameters are the parents and the (future) children of the crossover.

6.20.1 Detailed Description

Implementation of the generic crossover for the VRP-TW by Tavares et al.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 moeoVRPGenericCrossover::moeoVRPGenericCrossover ( ) `[inline]`

Deafult constructor.

6.20.3 Member Function Documentation

6.20.3.1 std::string moeoVRPGenericCrossover::className ( ) const `[inline]`

Returns a string containing the name of the class. Used to display statistics.

Returns

The string containing the name of the class.

6.20.3.2 bool moeoVRPGenericCrossover::operator() (moeoVRP & *_genotype1,* moeoVRP & *_genotype2*) [inline]

Both parameters are the parents and the (future) children of the crossover.

Parameters

_genotype1	The first parent.
_genotype2	The second parent.

Returns

True if any of the parents was modified. False otherwise.

Here is the call graph for this function:

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

- problems/VRPTW/moeoVRPQuadCrossover.h

6.21 moeoVRPInit Class Reference

Class defining the initializer functor. This class initializes an individual of the VRP problem using an heuristic initializer.

```
#include <moeoVRPInit.h>
```

Public Member Functions

- moeoVRPInit ()

  Default constructor: nothing to do here.

- void operator() (moeoVRP &_gen)

  Functor member. Initializes a genotype using an heuristic initializer.

6.21.1 Detailed Description

Class defining the initializer functor. This class initializes an individual of the VRP problem using an heuristic initializer.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 moeoVRPInit::moeoVRPInit ( ) [inline]

Default constructor: nothing to do here.

6.21.3 Member Function Documentation

6.21.3.1 void moeoVRPInit::operator() (moeoVRP & _gen) [inline]

Functor member. Initializes a genotype using an heuristic initializer.

Parameters

_gen	Generally a genotype that has been default-constructed. Whatever it contains will be lost.

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

- problems/VRPTW/moeoVRPInit.h

6.22 moeoVRPInsertionMutation Class Reference

Implementation of the insertion mutation operator.

```
#include <moeoVRPMutation.h>
```

Public Member Functions

- moeoVRPInsertionMutation ()

  Deafult constructor.

- std::string className () const

  Returns a string containing the name of the class. Used to display statistics.

- bool operator() (moeoVRP &_genotype)

  It selects and individual, erases it from its original position and inserts it somewhere else. The insertion may or may not be within the same route.

6.22.1 Detailed Description

Implementation of the insertion mutation operator.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 moeoVRPInsertionMutation::moeoVRPInsertionMutation ( ) [inline]

Deafult constructor.

6.22.3 Member Function Documentation

6.22.3.1 std::string moeoVRPInsertionMutation::className ( ) const [inline]

Returns a string containing the name of the class. Used to display statistics.

Returns

The string containing the name of the class.

6.22.3.2 bool moeoVRPInsertionMutation::operator() (moeoVRP & *_genotype*) [inline]

It selects and individual, erases it from its original position and inserts it somewhere else. The insertion may or may not be within the same route.

Parameters

_genotype	The genotype being mutated (it will be probably modified).

Returns

True if the individual has been modified. False otherwise.

Here is the call graph for this function:

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

- problems/VRPTW/moeoVRPMutation.h

6.23 moeoVRPInversionMutation Class Reference

Implementation of the inversion mutation operator.

`#include <moeoVRPMutation.h>`

Public Member Functions

- moeoVRPInversionMutation ()

  Deafult constructor.

- std::string className () const

  Returns a string containing the name of the class. Used to display statistics.

- bool operator() (moeoVRP &_genotype)

  It selects two positions in the genotype and inverts the clients between them. Clients may or may not be in the same route.

6.23.1 Detailed Description

Implementation of the inversion mutation operator.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 moeoVRPInversionMutation::moeoVRPInversionMutation ( ) `[inline]`

Deafult constructor.

6.23.3 Member Function Documentation

6.23.3.1 std::string moeoVRPInversionMutation::className ( ) const `[inline]`

Returns a string containing the name of the class. Used to display statistics.

Returns

The string containing the name of the class.

6.23.3.2 bool moeoVRPInversionMutation::operator() (moeoVRP & *_genotype*) `[inline]`

It selects two positions in the genotype and inverts the clients between them. Clients may or may not be in the same route.

Parameters

_genotype	The genotype being mutated (it will be probably modified).

Returns

True if the individual has been modified. False otherwise.

Here is the call graph for this function:

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

- problems/VRPTW/moeoVRPMutation.h

6.24 moeoVRPIterSwap< EOT > Class Template Reference

```
#include <moeoVRPIterSwap.h>
```

Inheritance diagram for moeoVRPIterSwap< EOT >:

Collaboration diagram for moeoVRPIterSwap< EOT >:

Public Member Functions

- moeoVRPIterSwap ()
- moeoVRPIterSwap (unsigned _i, unsigned _k)
- void updateParam (EOT &_routePlan)
- bool reset (EOT &_solution)
- void initParam (EOT &_routePlan)
- void undo (EOT &_routePlan)
- bool move (EOT &_routePlan)
- bool accept (EOT &_solution)
- void terminate (EOT &_solution)

template<class EOT> class moeoVRPIterSwap< EOT >

6.24.1 Constructor & Destructor Documentation

6.24.1.1 template<class EOT > moeoVRPIterSwap< EOT >::moeoVRPIterSwap ( ) [inline]

6.24.1.2 template<class EOT > moeoVRPIterSwap< EOT >::moeoVRPIterSwap (unsigned *_i*, unsigned *_k*) [inline]

6.24.2 Member Function Documentation

6.24.2.1 template<class EOT > bool moeoVRPIterSwap< EOT >::accept (EOT & *_solution*) [inline, virtual]

Implements moNeighborhoodExplorer< EOT >.

6.24.2.2 template<class EOT > void moeoVRPIterSwap< EOT >::initParam (EOT & *_routePlan*) [inline, virtual]

Implements moNeighborhoodExplorer< EOT >.

6.24.2.3 template<class EOT > bool moeoVRPIterSwap< EOT >::move (EOT & *_routePlan*) [inline, virtual]

Implements moNeighborhoodExplorer< EOT >.

Here is the call graph for this function:

6.24.2.4 template<class EOT > bool moeoVRPIterSwap< EOT >::reset (EOT & *_solution*) [inline, virtual]

Implements moNeighborhoodExplorer< EOT >.

6.24.2.5 template<class EOT > void moeoVRPIterSwap< EOT >::terminate (EOT & *_solution*) [inline, virtual]

Implements moNeighborhoodExplorer< EOT >.

6.24.2.6 template<class EOT > void moeoVRPIterSwap< EOT >::undo (EOT & *_routePlan*) [inline]

Here is the call graph for this function:

6.24.2.7 template<class EOT > void moeoVRPIterSwap< EOT >::updateParam (EOT & *_routePlan*) [inline, virtual]

Implements moNeighborhoodExplorer< EOT >.

Here is the caller graph for this function:

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

- problems/VRPTW/moeoVRPIterSwap.h

6.25 moeoVRPObjectiveVectorTraits Class Reference

```
#include <moeoVRPObjectiveVectorTraits.h>
```

Static Public Member Functions

- static bool minimizing (int i)
- static bool maximizing (int i)
- static unsigned int nObjectives ()

6.25.1 Member Function Documentation

6.25.1.1 static bool moeoVRPObjectiveVectorTraits::maximizing (int *i*) [inline, static]

6.25.1.2 static bool moeoVRPObjectiveVectorTraits::minimizing (int *i*) [inline, static]

6.25.1.3 static unsigned int moeoVRPObjectiveVectorTraits::nObjectives ( ) [inline, static]

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

- problems/VRPTW/moeoVRPObjectiveVectorTraits.h

6.26 moeoVRPOnePointCrossover Class Reference

Implementation of the simple One Point Crossover.

```
#include <moeoVRPQuadCrossover.h>
```

Public Member Functions

- moeoVRPOnePointCrossover ()

  Deafult constructor.

- std::string className () const

  Returns a string containing the name of the class. Used to display statistics.

- bool operator() (moeoVRP &_genotype1, moeoVRP &_genotype2)

  Performs a one point crossover. Both parameters are the parents and the (future) children of the crossover.

6.26.1 Detailed Description

Implementation of the simple One Point Crossover.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 moeoVRPOnePointCrossover::moeoVRPOnePointCrossover ( ) `[inline]`

Deafult constructor.

6.26.3 Member Function Documentation

6.26.3.1 std::string moeoVRPOnePointCrossover::className ( ) const `[inline]`

Returns a string containing the name of the class. Used to display statistics.

Returns

The string containing the name of the class.

Here is the caller graph for this function:

6.26.3.2 bool moeoVRPOnePointCrossover::operator() (moeoVRP & *_genotype1,* moeoVRP & *_genotype2*) `[inline]`

Performs a one point crossover. Both parameters are the parents and the (future) children of the crossover.

Parameters

_genotype1	The first parent.
_genotype2	The second parent.

Returns

True if any of the parents was modified. False otherwise.

Here is the call graph for this function:

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

- problems/VRPTW/moeoVRPQuadCrossover.h

6.27 moeoVRPStat Class Reference

Manages the statistics of the VRP problem.

`#include <moeoVRPStat.h>`

Public Member Functions

- moeoVRPStat (std::string _description="moeoVRPStat ")

  Constructor: initializes variables properly.

- void operator() (const eoPop< moeoVRP > &_pop)

  Gets statistics from a population.

- virtual std::string className (void) const

  Returns a string containing the name of the class. Used to display statistics.

6.27.1 Detailed Description

Manages the statistics of the VRP problem.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 moeoVRPStat::moeoVRPStat (std::string *_description* = "moeoVRPStat ") [inline]

Constructor: initializes variables properly.

Parameters

_description	A string identifying the class.

6.27.3 Member Function Documentation

6.27.3.1 virtual std::string moeoVRPStat::className (void) const [inline, virtual]

Returns a string containing the name of the class. Used to display statistics.

Returns

The string containing the name of the class.

6.27.3.2 void moeoVRPStat::operator() (const eoPop< moeoVRP > & _pop) [inline]

Gets statistics from a population.

Parameters

_pop	The population that will be analyzed.

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

- problems/VRPTW/moeoVRPStat.h

6.28 moeoVRPSwapMutation Class Reference

Implementation of the swap mutation operator.

```
#include <moeoVRPMutation.h>
```

Public Member Functions

- moeoVRPSwapMutation ()

  Deafult constructor.

- std::string className () const

  Returns a string containing the name of the class. Used to display statistics.

- bool operator() (moeoVRP &_genotype)

  It exhanges the positions of two clients within the individual. Clients may or may not be in the same route.

6.28.1 Detailed Description

Implementation of the swap mutation operator.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 moeoVRPSwapMutation::moeoVRPSwapMutation ( ) [inline]

Deafult constructor.

6.28.3 Member Function Documentation

6.28.3.1 std::string moeoVRPSwapMutation::className ( ) const [inline]

Returns a string containing the name of the class. Used to display statistics.

Returns

The string containing the name of the class.

6.28.3.2 bool moeoVRPSwapMutation::operator() (moeoVRP & *_genotype*) [inline]

It exhanges the positions of two clients within the individual. Clients may or may not be in the same route.

Parameters

_genotype	The genotype being mutated (it will be probably modified).

Returns

True if the individual has been modified. False otherwise.

Here is the call graph for this function:

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

- problems/VRPTW/moeoVRPMutation.h

6.29 moMOLS< EOT, tObjectiveVector > Class Template Reference

Variable Neighbors Search (VNS)

`#include <moMOLS.h>`

Collaboration diagram for moMOLS< EOT, tObjectiveVector >:

Public Member Functions

- moMOLS (moNeighborhoodExplorer< EOT > *_explorer, eoEvalFunc< EOT > &_evaluation)

Generic constructor.

- void setAcceptanceStrategy (const std::string &_criterion)
- bool firstImprovement () const
- bool bestImprovement () const
- bool randomImprovement () const
- bool operator() (EOT &_solution)

  Function which launches the VNS.

6.29.1 Detailed Description

template<class EOT, class tObjectiveVector> class moMOLS< EOT, tObjectiveVector >

Variable Neighbors Search (VNS) Class which describes the algorithm for a Variable Neighbors Search.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 template<class EOT, class tObjectiveVector> moMOLS< EOT, tObjectiveVector >::moMOLS (moNeighborhoodExplorer< EOT > ∗ *_explorer,* eoEvalFunc< EOT > & *_evaluation*) `[inline]`

Generic constructor.

Generic constructor using a moExpl

Parameters

_explorer	Vector of Neighborhoods.
full-evaluation	The evaluation function.

6.29.3 Member Function Documentation

6.29.3.1 template<class EOT, class tObjectiveVector> bool moMOLS< EOT, tObjectiveVector >::bestImprovement ( ) const `[inline]`

6.29.3.2 template<class EOT, class tObjectiveVector> bool moMOLS< EOT, tObjectiveVector >::firstImprovement ( ) const `[inline]`

Here is the caller graph for this function:

6.29.3.3 template<class EOT, class tObjectiveVector> bool moMOLS< EOT, tObjectiveVector >::operator() (EOT & _solution) [inline]

Function which launches the VNS.

The LS has to improve a current solution.

Parameters

_solution	a current solution to improve (we assume this is a multi-objective solution)

Returns

true if there was an improvement, false otherwise

Here is the call graph for this function:

6.29.3.4 template<class EOT, class tObjectiveVector> bool moMOLS< EOT, tObjectiveVector >::randomImprovement ( ) const [inline]

6.29.3.5 template<class EOT, class tObjectiveVector> void moMOLS< EOT, tObjectiveVector >::setAcceptanceStrategy (const std::string & _criterion) [inline]

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

- core/moMOLS.h

6.30 moNeighborhoodExplorer< EOT > Class Template Reference

`#include <moNeighborhoodExplorer.h>`

Inheritance diagram for moNeighborhoodExplorer< EOT >:

Public Member Functions

- virtual void initParam (EOT &_solution)=0
- virtual void updateParam (EOT &_solution)=0
- virtual bool reset (EOT &_solution)=0
- virtual bool move (EOT &_solution)=0
- virtual bool accept (EOT &_solution)=0
- virtual void terminate (EOT &_solution)=0

template<class EOT> class moNeighborhoodExplorer< EOT >

6.30.1 Member Function Documentation

6.30.1.1 template<class EOT> virtual bool moNeighborhoodExplorer< EOT >::accept (EOT & *_solution*) `[pure virtual]`

Implemented in moeoVRPIterSwap< EOT >.

6.30.1.2 template<class EOT> virtual void moNeighborhoodExplorer< EOT >::initParam (EOT & *_solution*) `[pure virtual]`

Implemented in moeoVRPIterSwap< EOT >.

6.30.1.3 template<class EOT> virtual bool moNeighborhoodExplorer< EOT >::move (EOT & *_solution*) `[pure virtual]`

Implemented in moeoVRPIterSwap< EOT >.

6.30.1.4 template<class EOT> virtual bool moNeighborhoodExplorer< EOT >::reset (EOT & *_solution*) `[pure virtual]`

Implemented in moeoVRPIterSwap< EOT >.

6.30.1.5 template<class EOT> virtual void moNeighborhoodExplorer< EOT >::terminate (EOT & *_solution*) `[pure virtual]`

Implemented in moeoVRPIterSwap< EOT >.

6.30.1.6 template<class EOT> virtual void moNeighborhoodExplorer< EOT >::updateParam (EOT & *_solution*) `[pure virtual]`

Implemented in moeoVRPIterSwap< EOT >.

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

- core/moNeighborhoodExplorer.h

6.31 neighborhood< EOT > Class Template Reference

Interface for creating neighborhoods.

`#include <neighborhood.h>`

Inheritance diagram for neighborhood< EOT >:

Public Member Functions

- neighborhood ()
- ∼neighborhood ()
- void setRecipients (eoPop< EOT ∗ > ∗_recipients)
- void setRecipients (eoPop< EOT > &_recipients)
- void addRecipient (agent< EOT > ∗recipient)
- virtual std::vector< agent< EOT > ∗ > ∗const list () const =0

Protected Member Functions

- std::vector< agent< EOT > ∗ > ∗const getRecipients () const

6.31.1 Detailed Description

template<class EOT> class neighborhood< EOT >

Interface for creating neighborhoods. This class acts as an interface between the agent class and user neighborhoods classes. This implements the attributes and methods needed to manage a wide variety of neighboring topologies.

From a 'state' point of view, we can discern two types of communication topologies:

- static.
- dynamic.

In order to optimize both topologies, this class has been conceived to store a pointer to the agents vector for dynamic topologies and a pointer to a vector with the agents for the current topology in case of static topologies.

This is due to with static topologies it is not necessary to change the neighbors list in real time. It is just needed to initialize a static list of neighbors for each agent. For dynamic neighborhoods, the pointer to the vector of agents, must be a pointer to the same vector ssystem counts with. This way, any type of dynamic neighborhood inherited from this class are not to change this vector, instead it will has its own data-structure. For example, suppose we are going to work with a 'scoreBasedNeighborhood' (SBN). This type of neighborhood will have access to the vector of agents whithin the system. For creating the SBN, it shouldn't try to modify this vector but, for example, it should store a vector of pairs <agentdIds, scores>. Using the phases, the agents would be able to update this vector of scores. At the appropiate time, when the list of neighbors are needed in a communication phase, using this data-structure the scoreBasedNeighborhoood has and the pointer to the vector of agents of ssystem, the new neighbors list would be computed.

This methodology allows not to have any computation for static neigborhoods and flexibility for dynamic ones in cases of creation or deletation of agents in real time.

Author

Group of Intelligent Computing - Universidad de La Laguna - 2008

6.31.2 Constructor & Destructor Documentation

6.31.2.1 template<class EOT> neighborhood< EOT >::neighborhood ( ) [inline]

Default constructor. It initilizes the agent's pointer to NULL.

6.31.2.2 template<class EOT> neighborhood< EOT >::∼neighborhood ( ) [inline]

Default constructor. It does nothing. This method is NOT supposed to destroy the memory in which the agents reside, but is ssystem that should perform this operation.

6.31.3 Member Function Documentation

6.31.3.1 template<class EOT> void neighborhood< EOT >::addRecipient (agent< EOT > ∗ *recipient*) [inline]

Method that adds an agent to the neighboring list.

Parameters

a	pointer to a new recipient.

Warning

This mehod is design for static neihgboring lists only.

6.31.3.2 template<class EOT> std::vector<agent<EOT>∗>∗ const neighborhood< EOT >::getRecipients ( ) const [inline, protected]

Method that returns a pointer to the list of agents in this class. This method is protected because it has been designed to be accessible only for inherited classes. The method that is inteded to return the neighboring list is a virtual method getNeighborhood (see more details bellow).

Returns

a pointer to the group of agents in this class.

Here is the caller graph for this function:

6.31.3.3 template<class EOT> virtual std::vector<agent<EOT>∗>∗ const neighborhood< EOT >::list ( ) const [pure virtual]

Virtual method that returns the neighboring list to send the message.

Returns

a pointer to the group of agent who will receive the message.

Implemented in staticNeighborhood< EOT >.

6.31.3.4 template<class EOT> void neighborhood< EOT >::setRecipients (eoPop< EOT > & *_recipients*) [inline]

6.31.3.5 template<class EOT> void neighborhood< EOT >::setRecipients (eoPop< EOT ∗ > ∗ *_recipients*) [inline]

Method that sets the neighboring list to send the message.

Parameters

a	pointer to the group of agents who will receive the message in case of static neighborhoods or to work with for dynamic neighborhoods

Here is the caller graph for this function:

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

- core/neighborhood.h

6.32 phase< EOT > Class Template Reference

Interface for the creation of phases.

`#include <phase.h>`

Inheritance diagram for phase< EOT >:

Public Member Functions

- phase ()
- virtual ∼phase ()
- virtual void operator() (EOT ∗, eoPop< EOT > &)=0
- virtual void operator() (eoPop< EOT > &)=0
- virtual phase< EOT > ∗ clone () const =0

6.32.1 Detailed Description

template<class EOT> class phase< EOT >

Interface for the creation of phases. This abstract class acts as an interface between agents and the operations they carry out. The agents are designed to run a number of phases repeteadly whithin the ssystem main-loop. The idea is that the main-loop calls iteratively a phase after phase without knowing which operation an agent is about to perform. In other words, this is just a way to provide a system in which each agent has a turn (phase) to do whatever operation he has to do.

This class is deliberately simple to enhace the creation of phases of all types as complex as the user wants.

This interface allows each agent to have a vector of different phases by using polymorphism.

Inherited classes must implement three classes. The idea behind this is to separate the code in three simple steps.

- prePhase: Operations to be performed before main operations (initialization)
- core: Operation this phase was designed for
- postPhase: Operations to be performad after main operations (update)

Author

Group of Intelligent Computing - Universidad de La Laguna - 2008

6.32.2 Constructor & Destructor Documentation

6.32.2.1 template<class EOT> phase< EOT >::phase ( ) `[inline]`

Default constructor. It does nothing.

6.32.2.2 template<class EOT> virtual phase< EOT >::~phase ( ) `[inline, virtual]`

Default destructor. It does nothing.

6.32.3 Member Function Documentation

6.32.3.1 template<class EOT> virtual phase<EOT>∗ phase< EOT >::clone ( ) const `[pure virtual]`

Implemented in defaultCommunicationPhase< EOT >, and moeoJFOPhaseAlgorithm< EOT >.

6.32.3.2 template<class EOT> virtual void phase< EOT >::operator() (EOT ∗ , eoPop< EOT > &) `[pure virtual]`

Method that is invoked from the ssystem's main-loop to start the phase. This simply calls the inherited methods explained above.

Implemented in defaultCommunicationPhase< EOT >, moeoJFOPhaseAlgorithm< EOT >, and dummyPhase< EOT >.

6.32.3.3 template<class EOT> virtual void phase< EOT >::operator() (eoPop< EOT > &) [pure virtual]

Implemented in defaultCommunicationPhase< EOT >, moeoJFOPhaseAlgorithm< EOT >, and dummyPhase< EOT >.

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

- core/phase.h

6.33 staticNeighborhood< EOT > Class Template Reference

To work with static neighborhoods.

```
#include <staticNeighborhood.h>
```

Inheritance diagram for staticNeighborhood< EOT >:

Collaboration diagram for staticNeighborhood< EOT >:

Public Member Functions

- staticNeighborhood ()
- staticNeighborhood (eoPop< EOT > &listOfRecipients)
- ∼staticNeighborhood ()
- std::vector< agent< EOT > ∗ > ∗const list () const

6.33.1 Detailed Description

template<class EOT> class staticNeighborhood< EOT >

To work with static neighborhoods. This class inherits from neighborhood to deal with static communication topologies. The idea is to create for each agent a vector of neihgbors. This vector is to be assigned to the vector in the neihgborhood class using the method setAgents contained in the same class. Thus, this clas will implement the virtual method getNeighborhood that will simply return the pointer to the vector contained in neihgborhood class.

Author

Group of Intelligent Computing - Universidad de La Laguna - 2008

6.33.2 Constructor & Destructor Documentation

6.33.2.1 template<class EOT > staticNeighborhood< EOT >::staticNeighborhood ( ) [inline]

Default constructor. It does nothing.

6.33.2.2 template<class EOT > staticNeighborhood< EOT >::staticNeighborhood (eoPop< EOT > & *listOfRecipients*) [inline]

Standard constructor.

Parameters

vector<agent	const is the vector that contains the agents this one is related to.

Returns

a static topology with the given data.

Here is the call graph for this function:

6.33.2.3 template<class EOT > staticNeighborhood< EOT >::~staticNeighborhood ( ) [inline]

Default destructor. It does nothing.

6.33.3 Member Function Documentation

6.33.3.1 template<class EOT > std::vector<agent<EOT>*>* const staticNeighborhood< EOT >::list ( ) const [inline, virtual]

Method that returns the list of recipients the agent is going to send its message

Returns

a pointer to the vector of recipients.

Implements neighborhood< EOT >.

Here is the call graph for this function:

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

- neighborhoods/staticNeighborhood.h

6.34 VRPSolution Class Reference

Wapper for std::queue that enables concurrent operatrions.

`#include <concurrentQueue.h>`

6.34.1 Detailed Description

Wapper for std::queue that enables concurrent operatrions. This class embends a std::queue object providing a thread-safe queue. To this aim, some classes borrowed from the boost library are used to control de access to the queue.

Author

Group of Intelligent Computing - Universidad de La Laguna - 2008

Class adapted from http://www.justsoftwaresolutions.co.uk/threading/implementing-a-thread

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

- libs/concurrentQueue.h

Chapter 7

File Documentation

7.1 agents/defaultCommunicationPhase.h File Reference

```
#include <vector>
#include <boost/shared_ptr.hpp>
#include <CODEATypes.h>
#include <phase.h>
```

Include dependency graph for defaultCommunicationPhase.h: This graph shows which files directly or indirectly include this file:

Classes

- class defaultCommunicationPhase< EOT >

    Carries out a by-default Communication Phase.

7.2 agents/moeoJFOPhaseAlgorithm.h File Reference

```
#include <memory>
#include "core/phase.h"
#include "core/eoSingleOp.h"
#include "core/moMOLS.h"
#include <algo/moeoEA.h>
#include <comparator/moeoStrictObjectiveVectorComparator.h>
#include <eoEvalFunc.h>
#include <eoGenOp.h>
```

```
#include <eoPop.h>
```

Include dependency graph for moeoJFOPhaseAlgorithm.h: This graph shows which files directly or indirectly include this file:

Classes

- class moeoJFOPhaseAlgorithm< EOT >

7.3 agents/moeoNSGAIIAgent.cpp File Reference

```
#include "moeoNSGAIIAgent.h"
```

Include dependency graph for moeoNSGAIIAgent.cpp:

7.4 core/agent.h File Reference

```
#include <deque>
#include <vector>
#include "container.h"
#include "CODEATypes.h"
#include "mailBox.h"
#include "neighborhood.h"
#include "phase.h"
#include "../libs/concurrentQueue.h"
```

Include dependency graph for agent.h: This graph shows which files directly or indirectly include this file:

Classes

- class agent< EOT >

  Contains propperties and methods agents work with at a high level.

Typedefs

- typedef std::vector< container > tMessage

7.4.1 Typedef Documentation

7.4.1.1 typedef std::vector<container> tMessage

7.5 core/CODEATypes.h File Reference

```
#include <vector>
#include <moeo>
#include "container.h"
```

Include dependency graph for CODEATypes.h: This graph shows which files directly or indirectly include this file:

Typedefs

- typedef std::vector< container > tMessage

7.5.1 Typedef Documentation

7.5.1.1 typedef std::vector<container> tMessage

7.6 core/container.h File Reference

```
#include <iostream>
#include <string>
#include <boost/any.hpp>
```

Include dependency graph for container.h: This graph shows which files directly or indirectly include this file:

Classes

- class container

  Allows to store a pair compound by an id and any type of information.

7.7 core/eoMonSingleGenOp.h File Reference

```
#include <eoOp.h>
#include <eoGenOp.h>
#include <eoProportionalCombinedOp.h>
```

Include dependency graph for eoMonSingleGenOp.h: This graph shows which files directly or indirectly include this file:

Classes

- class eoMonSingleGenOp< EOT >

7.8 core/eoQuadSingleGenOp.h File Reference

```
#include <eoOp.h>
#include <eoGenOp.h>
#include <eoProportionalCombinedOp.h>
```

Include dependency graph for eoQuadSingleGenOp.h: This graph shows which files directly or indirectly include this file:

Classes

- class eoQuadSingleGenOp< EOT >

7.9 core/eoSingleOp.h File Reference

```
#include <eoOpContainer.h>
#include "core/eoQuadSingleGenOp.h"
#include "core/eoMonSingleGenOp.h"
```

Include dependency graph for eoSingleOp.h: This graph shows which files directly or indirectly include this file:

Classes

- class eoSingleOp< EOT >

7.10 core/mailBox.h File Reference

```
#include "../libs/concurrentQueue.h"
#include "CODEATypes.h"
#include "../libs/concurrentQueue.h"
```

Include dependency graph for mailBox.h: This graph shows which files directly or indirectly include this file:

Classes

- class mailBox

7.11 core/moeoCoverageMetric.h File Reference

```
#include <comparator/moeoParetoObjectiveVectorComparator.h>
#include <comparator/moeoWeakObjectiveVectorComparator.h>
#include <metric/moeoMetric.h>
```

Include dependency graph for moeoCoverageMetric.h: This graph shows which files directly or indirectly include this file:

Classes

- class moeoCoverageMetric< ObjectiveVector >

7.12 core/moMOLS.h File Reference

```
#include <mo>
#include "../agents/moeoJFOPhaseAlgorithm.h"
```

Include dependency graph for moMOLS.h: This graph shows which files directly or indirectly include this file:

Classes

- class moMOLS< EOT, tObjectiveVector >

  Variable Neighbors Search (VNS)

7.13 core/moNeighborhoodExplorer.h File Reference

```
#include <eoFunctor.h>
```

Include dependency graph for moNeighborhoodExplorer.h: This graph shows which files directly or indirectly include this file:

Classes

- class moNeighborhoodExplorer< EOT >

7.14 core/neighborhood.h File Reference

```
#include <vector>
#include "agent.h"
#include "CODEATypes.h"
```

Include dependency graph for neighborhood.h: This graph shows which files directly or indirectly include this file:

Classes

- class neighborhood< EOT >

  Interface for creating neighborhoods.

7.15 core/phase.h File Reference

```
#include <vector>
#include <eoAlgo.h>
#include <agent.h>
```

Include dependency graph for phase.h: This graph shows which files directly or indirectly include this file:

Classes

- class phase< EOT >

  Interface for the creation of phases.

7.16 do_makes/make_neighborhood.h File Reference

```
#include <vector>
#include <eoPop.h>
#include <utils/eoRNG.h>
#include "core/neighborhood.h"
#include "neighborhoods/staticNeighborhood.h"
```

Include dependency graph for make_neighborhood.h: This graph shows which files directly or indirectly include this file:

Functions

- template<class EOT >
  void do_make_star_topology (eoPop< EOT > &_population)
- template<class EOT >
  void do_make_ring_topology (eoPop< EOT > &_population)
- template<class EOT >
  void do_make_krandom_topology (eoPop< EOT > &_population)

7.16.1 Function Documentation

7.16.1.1 template<class EOT > void do_make_krandom_topology (eoPop< EOT > & *_population*)

7.16.1.2 template<class EOT > void do_make_ring_topology (eoPop< EOT > & *_population*)

Here is the call graph for this function:

7.16.1.3 template<class EOT > void do_make_star_topology (eoPop< EOT > & *_population*)

Here is the caller graph for this function:

7.17 dummyPhase.h File Reference

```
#include <vector>
#include "core/agent.h"
#include "core/phase.h"
#include "fitness/moeoDominanceDepthFitnessAssignment.h"
```

Include dependency graph for dummyPhase.h: This graph shows which files directly or indirectly include this file:

Classes

- class dummyPhase< EOT >

7.18 libs/concurrentQueue.h File Reference

```
#include <queue>
#include <boost/thread/thread.hpp>
#include <boost/thread/mutex.hpp>
```

`#include <boost/thread/condition_variable.hpp>`

Include dependency graph for concurrentQueue.h: This graph shows which files directly or indirectly include this file:

Classes

- class concurrentQueue< Data >

7.19 libs/conversions.h File Reference

`#include <string>`

`#include <cassert>`

`#include <sstream>`

Include dependency graph for conversions.h: This graph shows which files directly or indirectly include this file:

Functions

- template<typename T >
  T fromStringTo (const std::string &inputString)

  Implements a number of functions to convers data-types.
- template<typename T >
  std::string somethingToString (const T &something)
- double atod (char ∗number)

7.19.1 Function Documentation

7.19.1.1 double atod (char ∗ *number*) `[inline]`

Here is the caller graph for this function:

7.19.1.2 template<typename T > T fromStringTo (const std::string & *inputString*) `[inline]`

Implements a number of functions to convers data-types.

conversions

This library is intended to have a list of function related to conversions.

Author

> Group of Intelligent Computing - Universidad de La Laguna - 2008 Function that receives a std::string and return a value converted to T.

Parameters

const	std::string& is a constant reference to the string value we want to convert.

Returns

T value with the given value converted.

Note

this is specially useful when we read plain-text files with data.

```
// This is just an example of its use.
std::string stringNumber = "120";
double doubleNumber = fromStringTo<double>(stringNumber);
```

7.19.1.3 template<typename T > std::string somethingToString (const T & *something*) [inline]

Function that receives a value (T) and return a std::string containing that value.

Parameters

const	T& is a constant reference to the value we want to convert.

Returns

std::string with the given value.

```
// This is just an example of its use.
double doubleNumber = 120.05;
std::string stringNumber = somethingToString<std::string>(doubleNumber);
```

7.20 main.cpp File Reference

```
#include <eo>
#include <moeo>
#include <mo>
#include <utils/eoRealBounds.h>
#include <vector>
#include <iomanip>
#include "problems/VRPTW/moeoVRPEvalFunc.h"
#include "problems/VRPTW/moeoVRP.h"
#include "problems/VRPTW/moeoVRPQuadCrossover.h"
#include "problems/VRPTW/moeoVRPMutation.h"
#include "problems/VRPTW/moeoVRPInit.h"
```

```
#include "problems/VRPTW/moeoVRPIterSwap.h"
#include "problems/VRPTW/do_makes.h"
#include "do_makes/make_neighborhood.h"
#include "do/make_checkpoint.h"
#include "core/moMOLS.h"
#include "agents/defaultCommunicationPhase.h"
#include "agents/moeoJFOPhaseAlgorithm.h"
#include "moeoCoverageMetric.h"
#include "dummyPhase.h"
#include "libs/conversions.h"
```

Include dependency graph for main.cpp:

Defines

- #define VERBOSE false

Functions

- template< class EOT >
  void runJFO (eoPop< EOT > &pop, moeoEvalFunc< EOT > &_eval, eoParser &_parser, eoState &_state, unsigned long _maxGenerations, std::vector< tObjectiveVector > &set)
- template< class EOT >
  void runNSGAII (eoPop< EOT > &_pop, eoCheckPoint< EOT > &_checkpoint, eoParser &_parser, eoState &_state, moeoEvalFunc< EOT > &_eval, std::vector< tObjectiveVector > &set)
- int main (int argc, char *argv[ ])

Variables

- uint32_t seedValue
- double c1
- double c2
- double c3
- double c4

7.20.1 Define Documentation

7.20.1.1 #define VERBOSE false

7.20.2 Function Documentation

7.20.2.1 int main (int *argc*, char $*$ *argv[]*)

Here is the call graph for this function:

7.20.2.2 template<class EOT > void runJFO (eoPop< EOT > & *pop*, moeoEvalFunc< EOT > & *_eval*, eoParser & *_parser*, eoState & *_state*, unsigned long *_maxGenerations*, std::vector< tObjectiveVector > & *set*)

Here is the call graph for this function:

7.20.2.3 template<class EOT > void runNSGAII (eoPop< EOT > & *_pop*, eoCheckPoint< EOT > & *_checkpoint*, eoParser & *_parser*, eoState & *_state*, moeoEvalFunc< EOT > & *_eval*, std::vector< tObjectiveVector > & *set*)

Here is the call graph for this function:

7.20.3 Variable Documentation

7.20.3.1 double c1

7.20.3.2 double c2

7.20.3.3 double c3

7.20.3.4 double c4

7.20.3.5 uint32_t seedValue

definition of evaluation: class moeoVRPEvalFunc MUST derive from eoEvalFunc<moeoVRP> and should test for validity before doing any computation see tutorial/Templates/evalFunc.tmpl definition of representation: class moeoVRP MUST derive from EO<FitT> for some fitness definitions of operators: write as many classes as types of operators and include them here. In this simple example, one crossover (2->2) and one mutation (1->1) operators are used

7.21 moeoStrictObjectiveVectorComparator.h File Reference

```
#include <comparator/moeoObjectiveVectorComparator.h>
```

Include dependency graph for moeoStrictObjectiveVectorComparator.h:

Classes

- class moeoStrictObjectiveVectorComparator< ObjectiveVector >

7.22 neighborhoods/staticNeighborhood.h File Reference

```
#include <vector>
#include <eoPop.h>
#include "core/neighborhood.h"
```

Include dependency graph for staticNeighborhood.h: This graph shows which files directly or indirectly include this file:

Classes

- class staticNeighborhood< EOT >

  To work with static neighborhoods.

7.23 problems/VRPTW/do_makes.h File Reference

```
#include <eo>
#include <moeo>
#include <do/make_continue.h>
#include "moeoVRP.h"
#include "moeoVRPQuadCrossover.h"
#include "moeoVRPMutation.h"
#include "core/eoSingleOp.h"
```

Include dependency graph for do_makes.h: This graph shows which files directly or indirectly include this file:

Functions

- template<class EOT >
  eoCheckPoint< EOT > & do_make_checkpoint (eoParser &_parser, eoState &_state, eoContinue< EOT > &_continue)
- eoContinue< moeoVRP > & do_make_continue (eoParser &_parser, eoState &_state)

Sets the stop criterion.

- eoGenOp< moeoVRP > & do_make_op (eoParser &_parser, eoState &_state, std::string _opt="")
- template<class EOT >
  eoPop< EOT > & do_make_pop (eoParser &_parser, eoState &_state, eoInit< EOT > &_init)

  Creates the population.
- template<class tOV >
  moeoObjectiveVectorComparator< tOV > & do_make_comparator ()

7.23.1 Function Documentation

7.23.1.1 template<class EOT > eoCheckPoint<EOT>& do_make_checkpoint (eoParser & *_parser,* eoState & *_state,* eoContinue< EOT > & *_continue*)

definition of representation: class moeoVRP MUST derive from EO<FitT> for some fitness

Here is the caller graph for this function:

7.23.1.2 template<class tOV > moeoObjectiveVectorComparator<tOV>& do_make_comparator ( )

7.23.1.3 eoContinue<moeoVRP>& do_make_continue (eoParser & *_parser,* eoState & *_state*)

Sets the stop criterion.

Parameters

_parser	Contains the parameters grabbed by the Paradiseo's parser
_state	Stores the object to set the stopping criterion

Here is the caller graph for this function:

7.23.1.4 eoGenOp<moeoVRP>& do_make_op (eoParser & *_parser,* eoState & *_state,* std::string *_opt* = " ")

This function builds the operators that will be applied to the moeoVRP

Parameters

eoParser&	_parser to get user parameters
eoState&	_state to store the memory

Here is the caller graph for this function:

7.23.1.5 template<class EOT > eoPop<EOT>& do_make_pop (eoParser & _parser, eoState & _state, eoInit< EOT > & _init)

Creates the population.

Parameters

_parser	Contains the parameters grab by the Paradiseo's eoParser
_state	Container to keep objects allocated.
_init	Initialisation of the chromosomes.

Here is the caller graph for this function:

7.24 problems/VRPTW/moeoVRP.h File Reference

```
#include <algorithm>
#include <vector>
#include <eoVector.h>
#include "moeoVRPUtils.h"
#include <limits>
#include "core/agent.h"
#include <string>
#include "moeoVRPObjectiveVectorTraits.h"
#include "core/CODEATypes.h"
```

Include dependency graph for moeoVRP.h: This graph shows which files directly or indirectly include this file:

Classes

- class moeoVRP

  Defines the getoype used to solve the VRP-TW problem. Objectives:

  - *<Size of="" the="" fleet>="">int is number of vehicles we need to use to satisfy all costumers. – <Travel distance>="">double is length of the route-plan. – <Travel time>="">double is elapsed time since the first delivery vehicle departs from the depot until the last arrived at the depot. – <Waiting time>="">double is the amount of time that vehicles have to wait at each costumer location. – <Delay time>="">double is the amount of time by which the arrival of the vehicles + service time is retarded respect to the closing time of the costumers.*

Defines

- #define INFd std::numeric_limits<double>::infinity()

- #define INFi std::numeric_limits<long>::infinity()
- #define DELAY_MAX 1800

Typedefs

- typedef moeoRealObjectiveVector< moeoVRPObjectiveVectorTraits > moeoVRPObjectiveVector
- typedef moeoRealObjectiveVector< moeoVRPObjectiveVectorTraits > ObjectiveVector
- typedef moeoVRPObjectiveVector tObjectiveVector
- typedef moeoVector< tObjectiveVector, double, double, int > tIndividual

7.24.1 Define Documentation

7.24.1.1 #define DELAY_MAX 1800

7.24.1.2 #define INFd std::numeric_limits<double>::infinity()

7.24.1.3 #define INFi std::numeric_limits<long>::infinity()

7.24.2 Typedef Documentation

7.24.2.1 typedef moeoRealObjectiveVector<moeoVRPObjectiveVectorTraits> moeoVRPObjectiveVector

7.24.2.2 typedef moeoRealObjectiveVector<moeoVRPObjectiveVectorTraits> ObjectiveVector

7.24.2.3 typedef moeoVector<tObjectiveVector, double, double, int> tIndividual

7.24.2.4 typedef moeoVRPObjectiveVector tObjectiveVector

7.25 problems/VRPTW/moeoVRPEvalFunc.h File Reference

```
#include <stdexcept>
#include <fstream>
#include "moeoVRPUtils.h"
#include "moeoVRPObjectiveVectorTraits.h"
#include "moeoVRP.h"
```

Include dependency graph for moeoVRPEvalFunc.h: This graph shows which files directly or indirectly include this file:

Classes

- class moeoVRPEvalFunc

Typedefs

- typedef moeoRealObjectiveVector< moeoVRPObjectiveVectorTraits > moeoVRPObjectiveVector

7.25.1 Typedef Documentation

7.25.1.1 typedef moeoRealObjectiveVector<moeoVRPObjectiveVectorTraits> moeoVRPObjectiveVector

7.26 problems/VRPTW/moeoVRPInit.h File Reference

```
#include <eoInit.h>
#include "moeoVRPUtils.h"
#include "moeoVRPObjectiveVectorTraits.h"
```

Include dependency graph for moeoVRPInit.h: This graph shows which files directly or indirectly include this file:

Classes

- class moeoVRPInit

  Class defining the initializer functor. This class initializes an individual of the VRP problem using an heuristic initializer.

Defines

- #define ALFA 0.7
- #define BETA 0.1
- #define GAMMA 0.2

Typedefs

- typedef moeoRealObjectiveVector< moeoVRPObjectiveVectorTraits > moeoVRPObjectiveVector

7.26.1 Define Documentation

7.26.1.1 #define ALFA 0.7

Constant used by "selectCheapestClient" method.

7.26.1.2 #define BETA 0.1

Constant used by "selectCheapestClient" method.

7.26.1.3 #define GAMMA 0.2

Constant used by "selectCheapestClient" method.

7.26.2 Typedef Documentation

7.26.2.1 typedef moeoRealObjectiveVector<moeoVRPObjectiveVectorTraits> moeoVRPObjectiveVector

7.27 problems/VRPTW/moeoVRPIterSwap.h File Reference

```
#include <mo>
#include "moNeighborhoodExplorer.h"
#include "moeoVRP.h"
```

Include dependency graph for moeoVRPIterSwap.h: This graph shows which files directly or indirectly include this file:

Classes

- class moeoVRPIterSwap< EOT >

7.28 problems/VRPTW/moeoVRPMutation.h File Reference

```
#include <algorithm>
#include <eoOp.h>
#include "moeoVRPUtils.h"
```

Include dependency graph for moeoVRPMutation.h: This graph shows which files directly or indirectly include this file:

Classes

- class moeoVRPSwapMutation

  Implementation of the swap mutation operator.

- class moeoVRPInsertionMutation

  Implementation of the insertion mutation operator.

- class moeoVRPInversionMutation

  Implementation of the inversion mutation operator.

- class moeoVRPDisplacementMutation

  Implementation of the displacement mutation operator.

7.29 problems/VRPTW/moeoVRPObjectiveVectorTraits.h File Reference

```
#include <moeo>
```

Include dependency graph for moeoVRPObjectiveVectorTraits.h: This graph shows which files directly or indirectly include this file:

Classes

- class moeoVRPObjectiveVectorTraits

7.30 problems/VRPTW/moeoVRPQuadCrossover.h File Reference

```
#include <cassert>
#include <climits>
#include <utils/eoRNG.h>
#include <set>
#include <eoOp.h>
#include "moeoVRPUtils.h"
```

Include dependency graph for moeoVRPQuadCrossover.h: This graph shows which files directly or indirectly include this file:

Classes

- class moeoVRPGenericCrossover

Implementation of the generic crossover for the VRP-TW by Tavares et al.

- class moeoVRPOnePointCrossover

  Implementation of the simple One Point Crossover.

- class moeoVRPEdgeCrossover

  Implementation of the classic Edge Crossover from the TSP.

7.31 problems/VRPTW/moeoVRPStat.h File Reference

`#include <utils/eoStat.h>`

Include dependency graph for moeoVRPStat.h:

Classes

- class moeoVRPStat

  Manages the statistics of the VRP problem.

7.32 problems/VRPTW/moeoVRPUtils.h File Reference

```
#include <cassert>
#include <cstdlib>
#include <vector>
#include <utility>
#include <fstream>
#include <iostream>
#include <sstream>
#include <math.h>
```

Include dependency graph for moeoVRPUtils.h: This graph shows which files directly or indirectly include this file:

Classes

- struct moeoVRPUtils::ClientData

  Information regarding each client in the dataset. This structure is intended to be used to store the information of each client read from the data file.

Namespaces

- namespace moeoVRPUtils

    A set of structures and utility functions for the VRP-TW problem.

Defines

- #define PI 3.14159265

Typedefs

- typedef std::vector< int > Route
- typedef std::vector< Route > Routes
- typedef struct moeoVRPUtils::ClientData moeoVRPUtils::ClientDataT

    Renaming of struct ClientData.

Functions

- void moeoVRPUtils::computeDistances ()

    Computes the distance between two clients. The computed distances will be stored in dist.

- void moeoVRPUtils::setTimeMatrixAsDistanceMatrix ()

    Set the time matrix to the distance matrix. This is needed in some datasets as they don't provide time information.

- void moeoVRPUtils::getTimeWindow (unsigned _client, double &_readyTime, double &_dueTime, double &_serviceTime)

    Returns the time window information of a given client.

- double moeoVRPUtils::distance (unsigned _from, unsigned _to)

    A function to get the distance between two clients.

- double moeoVRPUtils::elapsedTime (unsigned _from, unsigned _to)

    A function to get the distance between two clients.

- float moeoVRPUtils::polarAngle (unsigned _from, unsigned _to)

    Computes de polar angle between clients.

- void moeoVRPUtils::loadDistanceMatrix (const char ∗_filename)

    Loads the problem distance matrix data from a given file.

- void moeoVRPUtils::loadTimeMatrix (const char ∗_filename)

Loads the problem time matrix data from a given file.

- void moeoVRPUtils::load (const char ∗_fileName)

  Loads the problem data from a given file.

- void moeoVRPUtils::printRoute (const Route &_route)

  Prints a route to the standard output.

- void moeoVRPUtils::printRoutes (const Routes &_routes)

  Prints a set of routes to the standard output.

- void moeoVRPUtils::swap (Route &_routePlan, unsigned i, unsigned j)

  Swaps the position of two costumers.

- double moeoVRPUtils::travelDistance (const std::vector< int > &_routePlan)

  Evaluate the travel distance of a route plan.

- bool moeoVRPUtils::feasibleCapacity (const Route &_routePlan)

  Checks if a routePlan is feasible in terms of capacity.

- bool moeoVRPUtils::elapsedTimeBetweenTwoCostumers (double &_totalElapsedTime, unsigned &_numberOfViolations, unsigned _i, unsigned _j)

  Calculates the time that takes from a costumerto a costumer<j>.

- bool moeoVRPUtils::feasibleTimeWindows (const Route &_routePlan)

  Checks if a routePlan is feasible in terms of time windows.

- bool moeoVRPUtils::safetyCheck (const Route &_routePlan, std::string methodName="")
- bool moeoVRPUtils::safetyCheck (const std::vector< std::vector< int > > &_routePlan, std::string methodName="")
- void moeoVRPUtils::print ()

Variables

- double vehicleCapacity = 200
- unsigned sizeOfFleet = 0
- long moeoVRPUtils::evaluations
- long moeoVRPUtils::numberOfOperations

7.32.1 Define Documentation

7.32.1.1 #define PI 3.14159265

Guess you know what this constant represents.

7.32.2 Typedef Documentation

7.32.2.1 typedef std::vector<int> Route

7.32.2.2 typedef std::vector< Route > Routes

7.32.3 Variable Documentation

7.32.3.1 unsigned sizeOfFleet = 0

Max number of usable vehicles

7.32.3.2 double vehicleCapacity = 200

Max capacity of each vehicle within the fleet

7.33 VRPInstanceLoader.h File Reference

```
#include "problems/VRPTW/moeoVRPUtils.h"
#include "eoParser.h"
```

Include dependency graph for VRPInstanceLoader.h:

Functions

- void make_help (eoParser &_parser)
- void loadInstance (int argc, char ∗argv[], eoParser &parser)

7.33.1 Function Documentation

7.33.1.1 void loadInstance (int *argc,* char ∗ *argv[ ],* eoParser & *parser*)

Here is the call graph for this function:

7.33.1.2 void make_help (eoParser & *_parser*)