VRP Tabu Search

Generated by Doxygen 1.8.11

Contents

1 Class Index **1**

1.1 Class List . . . 1

2 File Index **3**

2.1 File List . . . 3

3 Class Documentation **5**

3.1 Edge::ConstructionToken Class Reference . . . 5

3.1.1 Constructor & Destructor Documentation . . . 5

3.1.1.1 ConstructionToken() . . . 5

3.1.2 Friends And Related Function Documentation . . . 5

3.1.2.1 Vertex . . . 5

3.2 Vertex::ConstructionToken Class Reference . . . 5

3.2.1 Constructor & Destructor Documentation . . . 6

3.2.1.1 ConstructionToken()=default . . . 6

3.2.2 Friends And Related Function Documentation . . . 6

3.2.2.1 Graph . . . 6

3.3 Controller Class Reference . . . 6

3.3.1 Constructor & Destructor Documentation . . . 7

3.3.1.1 Controller() . . . 7

3.3.1.2 Controller(Controller const &)=delete . . . 8

3.3.2 Member Function Documentation . . . 8

3.3.2.1 GetUtils() const . . . 8

3.3.2.2 Init(int, char ∗∗argv, float, float, int) . . . 8

3.3.2.3 Instance() 10
3.3.2.4 operator=(Controller const &)=delete 11
3.3.2.5 PrintBestRoutes() 11
3.3.2.6 PrintRoutes() 11
3.3.2.7 RunTabuSearch(int) 12
3.3.2.8 RunVRP() 13
3.3.2.9 SaveResult() 14
3.3.3 Member Data Documentation 15
3.3.3.1 finalCost 15
3.3.3.2 initCost 15
3.3.3.3 MAX_TIME_MIN 15
3.3.3.4 startTime 15
3.3.3.5 vrp 15
3.4 Customer Class Reference 15
3.4.1 Constructor & Destructor Documentation 16
3.4.1.1 Customer() 16
3.4.1.2 Customer(std::string n, int x, int y, int r, int t) 16
3.4.1.3 Customer(std::string n, int x, int y) 16
3.4.1.4 ∼Customer() 17
3.4.2 Member Function Documentation 17
3.4.2.1 operator"!=(const Customer &c) const 17
3.4.2.2 operator==(const Customer &c) const 17
3.4.3 Friends And Related Function Documentation 17
3.4.3.1 operator< 17
3.4.3.2 operator<< 17
3.4.4 Member Data Documentation 17
3.4.4.1 name 17
3.4.4.2 request 17
3.4.4.3 serviceTime 18
3.4.4.4 x 18

3.4.4.5 y . . . 18
3.5 Edge Class Reference . . . 18
3.5.1 Constructor & Destructor Documentation . . . 18
3.5.1.1 Edge(const Edge &) . . . 18
3.5.1.2 Edge(const ConstructionToken &, int) . . . 18
3.5.2 Member Data Documentation . . . 19
3.5.2.1 weight . . . 19
3.6 Graph Class Reference . . . 19
3.6.1 Constructor & Destructor Documentation . . . 20
3.6.1.1 Graph()=default . . . 20
3.6.2 Member Function Documentation . . . 20
3.6.2.1 GetCosts(const Customer &, const Customer &) const . . . 20
3.6.2.2 GetNeighborhood(Customer) const . . . 21
3.6.2.3 InsertEdge(Customer &, Customer &, int) . . . 22
3.6.2.4 InsertVertex(Customer &) . . . 22
3.6.2.5 InsertVertex(Customer &, Vertex &) . . . 23
3.6.2.6 RemoveEdge(Customer &, Customer &) . . . 23
3.6.2.7 sortV0() . . . 23
3.6.3 Member Data Documentation . . . 24
3.6.3.1 vertexes . . . 24
3.7 OptimalMove Class Reference . . . 24
3.7.1 Constructor & Destructor Documentation . . . 25
3.7.1.1 OptimalMove() . . . 26
3.7.2 Member Function Documentation . . . 26
3.7.2.1 AddRemoveFromTo(Route &, Route &, int, int) . . . 26
3.7.2.2 CleanVoid(Routes &) . . . 27
3.7.2.3 Move1FromTo(Route &, Route &, bool) . . . 28
3.7.2.4 Opt01(Routes &, bool) . . . 29
3.7.2.5 Opt10(Routes &, bool) . . . 30
3.7.2.6 Opt11(Routes &, bool) . . . 30

3.7.2.7 Opt12(Routes &, bool) 31
3.7.2.8 Opt2(Routes &) 32
3.7.2.9 Opt21(Routes &, bool) 33
3.7.2.10 Opt22(Routes &, bool) 33
3.7.2.11 Opt2Swap(Route, Customer, Customer) 34
3.7.2.12 Opt3(Routes &) 35
3.7.2.13 Opt3Swap(Route, Customer, Customer, Customer, Customer) 36
3.7.2.14 RouteBalancer(Routes &) 37
3.7.2.15 SwapFromTo(Route &, Route &) 37
3.7.2.16 UpdateDistanceAverage(Routes) 38
3.7.3 Member Data Documentation 38
3.7.3.1 cores 38
3.7.3.2 mtx 38
3.8 Route Class Reference 39
3.8.1 Constructor & Destructor Documentation 41
3.8.1.1 Route(const int, const float, const Graph, const float, const float) 41
3.8.2 Member Function Documentation 41
3.8.2.1 AddElem(const Customer) 41
3.8.2.2 AddElem(const std::list< Customer > &) 42
3.8.2.3 AddElem(const Customer, const Customer) 43
3.8.2.4 CloseTravel(const Customer) 43
3.8.2.5 CloseTravel(const Customer, const Customer) 44
3.8.2.6 CopyRoute() const 45
3.8.2.7 EmptyRoute(const Customer) 45
3.8.2.8 Evaluate() 45
3.8.2.9 FindCustomer(const Customer) 46
3.8.2.10 GetAverageCost() const 47
3.8.2.11 GetDistanceFrom(Route) 47
3.8.2.12 GetRoute() 48
3.8.2.13 GetTotalCost() const 48

3.8.2.14 GetUnderAverageCustomers(std::list< Customer > &) 48
3.8.2.15 operator"!=(const Route &r) const 49
3.8.2.16 operator<(const Route &r) const 49
3.8.2.17 operator<=(const Route &r) const 49
3.8.2.18 operator=(const Route &r) 50
3.8.2.19 operator==(const Route &r) const 50
3.8.2.20 PrintRoute() const 50
3.8.2.21 RebuildRoute(std::list< Customer >) 51
3.8.2.22 RemoveCustomer(std::list< StepType >::iterator &) 51
3.8.2.23 RemoveCustomer(const Customer) 52
3.8.2.24 SetAverageCost() 53
3.8.2.25 size() const 53
3.8.2.26 Travel(const Customer, const Customer) 53
3.8.3 Friends And Related Function Documentation 54
3.8.3.1 operator<< 54
3.8.4 Member Data Documentation 54
3.8.4.1 ALPHA 54
3.8.4.2 averageCost 54
3.8.4.3 capacity 55
3.8.4.4 graph 55
3.8.4.5 initialCapacity 55
3.8.4.6 initialWorkTime 55
3.8.4.7 route 55
3.8.4.8 totalCost 55
3.8.4.9 TRAVEL_COST 55
3.8.4.10 workTime 55
3.9 TabuList Class Reference 55
3.9.1 Constructor & Destructor Documentation 56
3.9.1.1 TabuList() 56
3.9.1.2 TabuList(int n) 57

3.9.2 Member Function Documentation 57
3.9.2.1 AddTabu(Move, int) 57
3.9.2.2 Check(Move) const 58
3.9.2.3 Clean() 58
3.9.2.4 DecrementSize() 59
3.9.2.5 Find(Move) const 59
3.9.2.6 FlushTabu() 59
3.9.2.7 IncrementSize() 59
3.9.2.8 RemoveTabu(Move) 60
3.9.3 Member Data Documentation 60
3.9.3.1 bestMoves 60
3.9.3.2 size 60
3.9.3.3 tabulist 60
3.10 TabuSearch Class Reference 60
3.10.1 Constructor & Destructor Documentation 61
3.10.1.1 TabuSearch(const Graph &g, const int n) 62
3.10.2 Member Function Documentation 62
3.10.2.1 Evaluate(Routes) 62
3.10.2.2 FindRouteFromCustomer(Customer, Routes) 62
3.10.2.3 Tabu(Routes &, int) 63
3.10.3 Member Data Documentation 64
3.10.3.1 fitness 64
3.10.3.2 graph 64
3.10.3.3 lambda 64
3.10.3.4 numCustomers 64
3.10.3.5 tabulist 64
3.11 ThreadPool Class Reference 64
3.11.1 Constructor & Destructor Documentation 65
3.11.1.1 ThreadPool(int c) 65
3.11.1.2 ∼ThreadPool() 65

3.11.2 Member Function Documentation 66
3.11.2.1 AddTask(std::function< void(void)> job) 66
3.11.2.2 JoinAll() 66
3.11.2.3 Run() 66
3.11.3 Member Data Documentation 67
3.11.3.1 queue 67
3.11.3.2 queue_mutex 67
3.11.3.3 stop 67
3.11.3.4 threadCount 67
3.11.3.5 threads 67
3.11.3.6 wait_var 67
3.12 Utils Class Reference 67
3.12.1 Constructor & Destructor Documentation 68
3.12.1.1 Utils() 68
3.12.1.2 Utils(Utils const &)=delete 68
3.12.2 Member Function Documentation 68
3.12.2.1 InitParameters(int, char ∗∗, const float, const float) 68
3.12.2.2 Instance() 70
3.12.2.3 logger(T s, int c=5) const 70
3.12.2.4 operator=(Utils const &)=delete 71
3.12.2.5 SaveResult(const std::list< Route >, int) 71
3.12.3 Member Data Documentation 72
3.12.3.1 ANSI_BLUE 72
3.12.3.2 ANSI_GREEN 72
3.12.3.3 ANSI_IBLUE 72
3.12.3.4 ANSI_LIGHTGREEN 72
3.12.3.5 ANSI_RED 72
3.12.3.6 ANSI_RESET 72
3.12.3.7 ANSI_YELLOW 72
3.12.3.8 d 72

3.12.3.9 ERROR 72
3.12.3.10 filename 72
3.12.3.11 INFO 72
3.12.3.12 SUCCESS 72
3.12.3.13 VERBOSE 73
3.12.3.14 verbose 73
3.12.3.15 WARNING 73
3.13 Vertex Class Reference 73
3.13.1 Constructor & Destructor Documentation 73
3.13.1.1 Vertex(ConstructionToken &) 73
3.13.2 Member Function Documentation 74
3.13.2.1 GetEdges() const 74
3.13.2.2 GetWeight(Customer &) 74
3.13.2.3 InsertEdge(Customer &, int) 74
3.13.2.4 RemoveEdge(Customer &) 75
3.13.3 Member Data Documentation 75
3.13.3.1 edges 75
3.14 VRP Class Reference 75
3.14.1 Constructor & Destructor Documentation 76
3.14.1.1 VRP() 77
3.14.1.2 VRP(const Graph &, const int, const int, const int, const float, const bool, const float, const float) 77
3.14.1.3 ∼VRP() 78
3.14.2 Member Function Documentation 78
3.14.2.1 CheckIntegrity() 78
3.14.2.2 CreateBest(std::set< Customer >, std::list< int >, Customer) 78
3.14.2.3 GetBestRoutes() 79
3.14.2.4 GetNumberOfCustomers() const 80
3.14.2.5 GetRoutes() 80
3.14.2.6 GetTotalCost() 80
3.14.2.7 init(int) 81

3.14.2.8 InitSolutions() 82
3.14.2.9 InitSolutionsNeigh() 82
3.14.2.10 OrderByCosts() 83
3.14.2.11 RunOpts(int, bool) 84
3.14.2.12 RunTabuSearch(int) 86
3.14.2.13 UpdateBest() 86
3.14.3 Member Data Documentation 87
3.14.3.1 alphaParam 87
3.14.3.2 averageDistance 87
3.14.3.3 bestRoutes 87
3.14.3.4 capacity 87
3.14.3.5 costTravel 87
3.14.3.6 graph 87
3.14.3.7 numVertices 87
3.14.3.8 routes 87
3.14.3.9 totalCost 88
3.14.3.10 vehicles 88
3.14.3.11 workTime 88

4 File Documentation 89

4.1 actor/Controller.cpp File Reference 89
4.2 actor/Controller.h File Reference 89
4.3 actor/Customer.cpp File Reference 90
4.4 actor/Customer.h File Reference 90
4.5 actor/OptimalMove.cpp File Reference 92
4.6 actor/OptimalMove.h File Reference 92
4.6.1 Typedef Documentation 93
4.6.1.1 BestResult 93
4.6.1.2 Routes 93
4.6.2 Variable Documentation 93
4.6.2.1 comp 93

4.7 actor/Route.cpp File Reference 94
4.8 actor/Route.h File Reference 94
4.8.1 Typedef Documentation 96
4.8.1.1 StepType 96
4.9 actor/TabuList.cpp File Reference 96
4.10 actor/TabuList.h File Reference 97
4.10.1 Typedef Documentation 98
4.10.1.1 Move 98
4.10.1.2 TabuElement 98
4.11 actor/TabuSearch.cpp File Reference 98
4.12 actor/TabuSearch.h File Reference 99
4.12.1 Typedef Documentation 101
4.12.1.1 RouteList 101
4.12.1.2 Routes 101
4.13 lib/Edge.cpp File Reference 101
4.14 lib/Edge.h File Reference 102
4.15 lib/Graph.cpp File Reference 103
4.16 lib/Graph.h File Reference 103
4.17 lib/ThreadPool.h File Reference 105
4.18 lib/Utils.cpp File Reference 106
4.19 lib/Utils.h File Reference 106
4.20 lib/Vertex.cpp File Reference 107
4.21 lib/Vertex.h File Reference 108
4.22 lib/VRP.cpp File Reference 110
4.23 lib/VRP.h File Reference 110
4.23.1 Typedef Documentation 111
4.23.1.1 Map 111
4.24 main.cpp File Reference 111
4.24.1 Macro Definition Documentation 112
4.24.1.1 ALPHA 112
4.24.1.2 MAX_TIME_MIN 112
4.24.1.3 TRAVEL_COST 112
4.24.2 Function Documentation 112
4.24.2.1 main(int argc, char **argv) 113

Chapter 1

Class Index

1.1 Class List

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

Edge::ConstructionToken 5
Vertex::ConstructionToken 5
Controller 6
Customer 15
Edge 18
Graph 19
OptimalMove 24
Route 39
TabuList 55
TabuSearch 60
ThreadPool 64
Utils 67
Vertex 73
VRP 75

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

main.cpp 111
actor/Controller.cpp 89
actor/Controller.h 89
actor/Customer.cpp 90
actor/Customer.h 90
actor/Route.cpp 94
actor/Route.h 94
actor/TabuList.cpp 96
actor/TabuList.h 97
actor/TabuSearch.cpp 98
actor/TabuSearch.h 99
lib/Edge.cpp 101
lib/Edge.h 102
lib/Graph.cpp 103
lib/Graph.h 103
lib/OptimalMove.cpp 92
lib/OptimalMove.h 92
lib/ThreadPool.h 105
lib/Utils.cpp 106
lib/Utils.h 106
lib/Vertex.cpp 107
lib/Vertex.h 108
lib/VRP.cpp 110
lib/VRP.h 110

Chapter 3

Class Documentation

3.1 Edge::ConstructionToken Class Reference

```
#include <Edge.h>
```

Private Member Functions

- ConstructionToken ()

Friends

- class Vertex

3.1.1 Constructor & Destructor Documentation

3.1.1.1 Edge::ConstructionToken::ConstructionToken ( ) `[private], [default]`

3.1.2 Friends And Related Function Documentation

3.1.2.1 friend class Vertex `[friend]`

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

- lib/Edge.h
- lib/Edge.cpp

3.2 Vertex::ConstructionToken Class Reference

```
#include <Vertex.h>
```

Private Member Functions

- ConstructionToken ()=default

Friends

- class Graph

3.2.1 Constructor & Destructor Documentation

3.2.1.1 Vertex::ConstructionToken::ConstructionToken ( ) [private], [default]

3.2.2 Friends And Related Function Documentation

3.2.2.1 friend class Graph [friend]

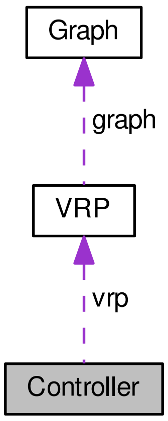
The documentation for this class was generated from the following file:

- lib/Vertex.h

3.3 Controller Class Reference

```
#include <Controller.h>
```

Collaboration diagram for Controller:



Public Member Functions

- void Init (int, char ∗∗argv, float, float, int)

  Configure variable and routes
- void RunVRP ()

  Runs all the main functions
- void SaveResult ()

  Save results.
- void PrintRoutes ()

  Print all routes.
- void PrintBestRoutes ()

  Print the best solution.
- Utils & GetUtils () const

Static Public Member Functions

- static Controller & Instance ()

Private Member Functions

- Controller ()
- Controller (Controller const &)=delete
- void operator= (Controller const &)=delete
- int RunTabuSearch (int)

  Runs the tabu search function.

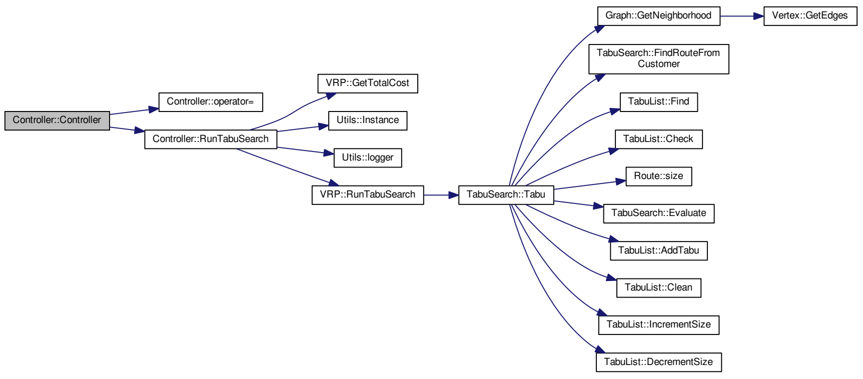
Private Attributes

- VRP ∗ vrp
- int MAX_TIME_MIN
- int initCost
- int finalCost
- std::chrono::high_resolution_clock::time_point startTime

3.3.1 Constructor & Destructor Documentation

3.3.1.1 Controller::Controller ( ) [inline], [private]

Here is the call graph for this function:

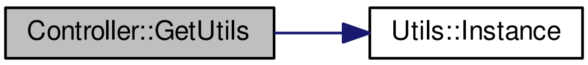


3.3.1.2 Controller::Controller (Controller const &) [private], [delete]

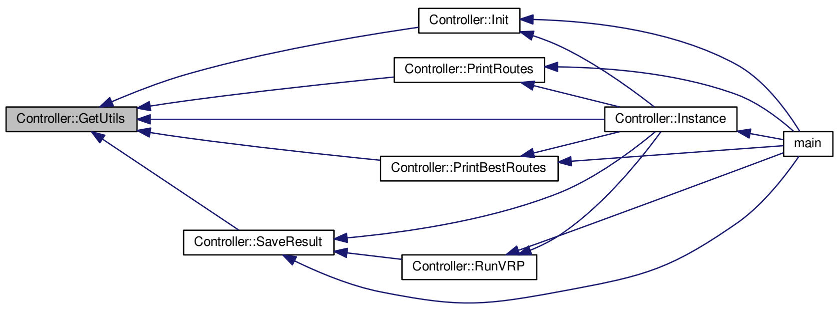
3.3.2 Member Function Documentation

3.3.2.1 Utils & Controller::GetUtils ( ) const

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.2.2 void Controller::Init (int *argc*, char ** *argv*, float *costTravel*, float *alphaParam*, int *max_time*)

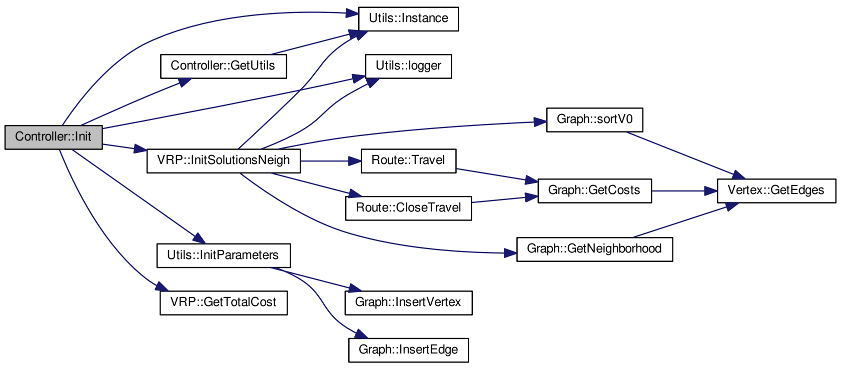
Configure variable and routes

The controller start the program settings all the variable and calling the functions to configure the routes.

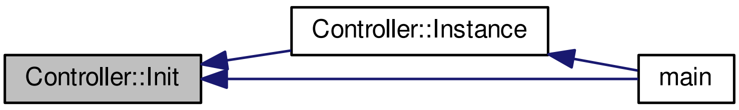
Parameters

in	*argc*	The number of arguments passed through command line.
in	*argv*	The arguments passed through command line.
in	*costTravel*	The cost of travelling.
in	*alphaParam*	Alpha parameter for route evaluation.
in	*max_time*	Maximum execution time in minutes.

Here is the call graph for this function:

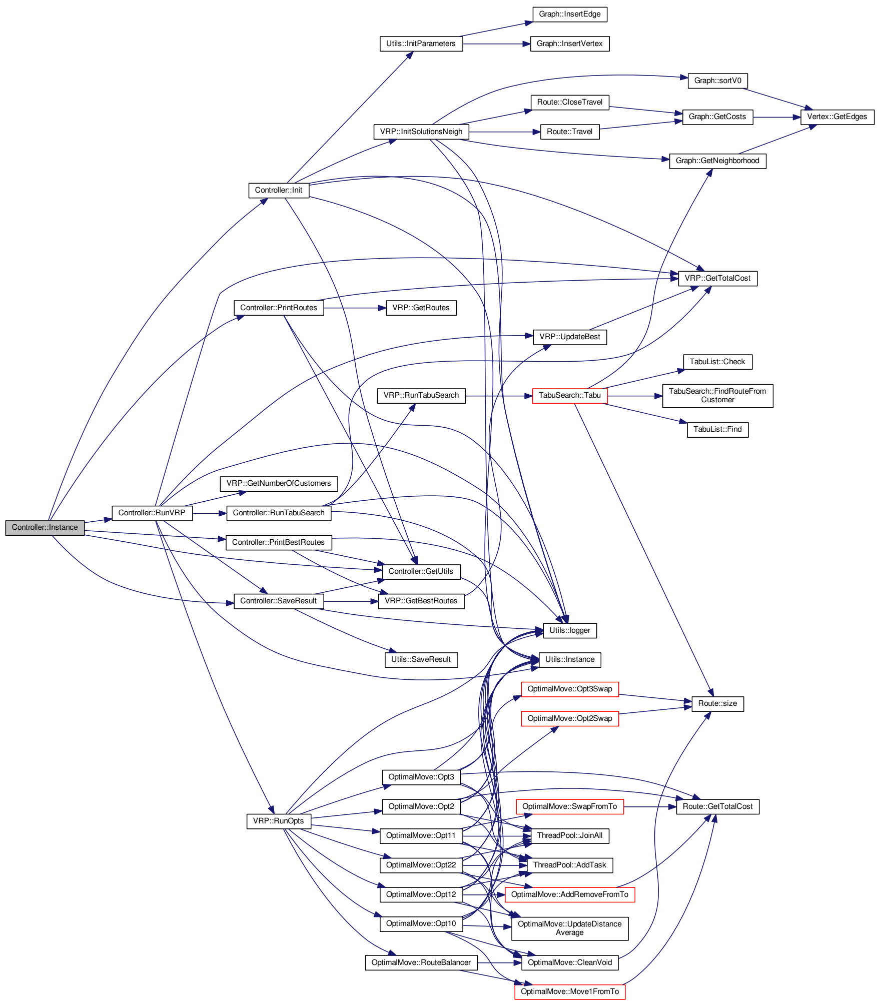


Here is the caller graph for this function:

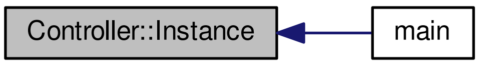


3.3.2.3 static Controller& Controller::Instance ( ) [inline], [static]

Here is the call graph for this function:

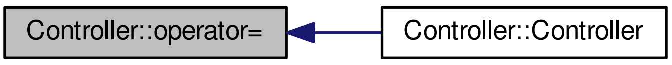


Here is the caller graph for this function:



3.3.2.4 void Controller::operator= (Controller const &) [private], [delete]

Here is the caller graph for this function:

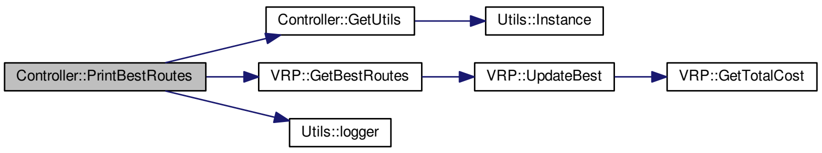


3.3.2.5 void Controller::PrintBestRoutes ( )

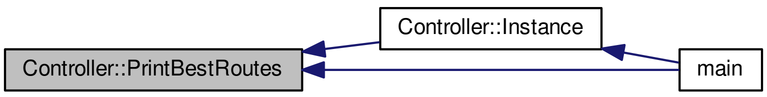
Print the best solution.

Prints all routes with costs and in a more readable way.

Here is the call graph for this function:



Here is the caller graph for this function:

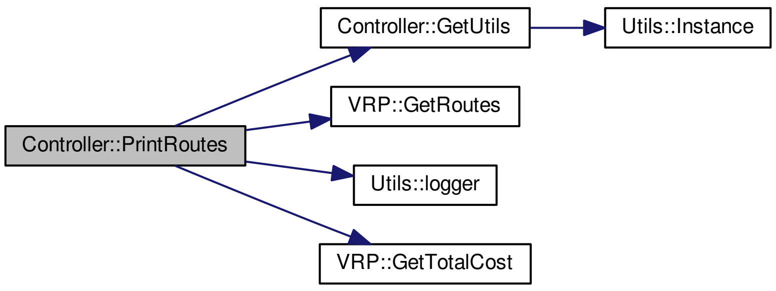


3.3.2.6 void Controller::PrintRoutes ( )

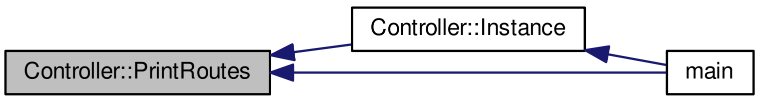
Print all routes.

Prints all routes with costs and in a more readable way.

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.2.7 int Controller::RunTabuSearch (int *times*) `[private]`

Runs the tabu search function.

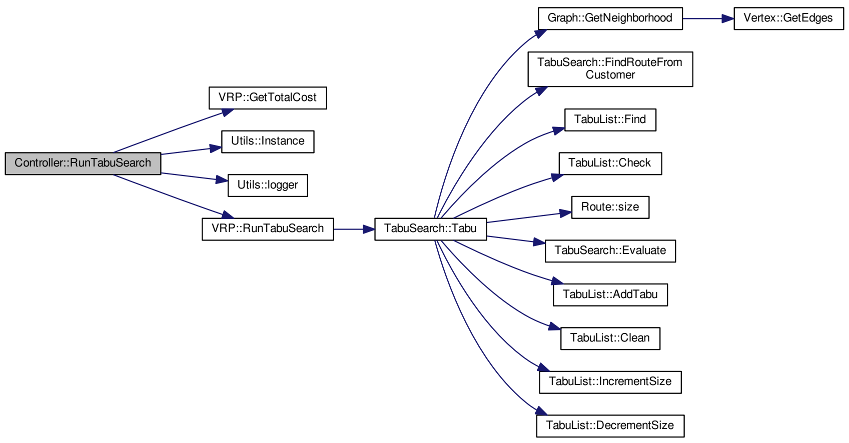
Parameters

`in`	*times*	Number of iteration.

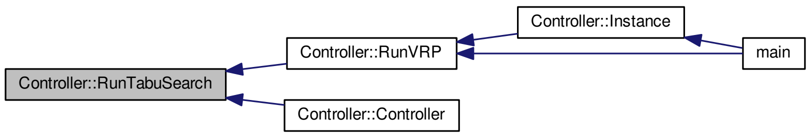
Returns

If the routine made some improvements.

Here is the call graph for this function:



Here is the caller graph for this function:

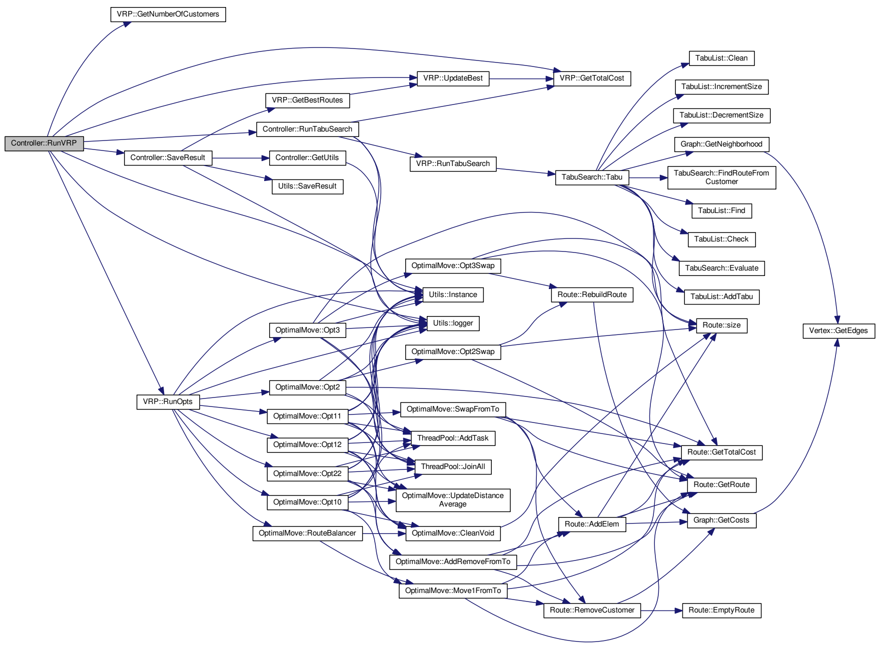


3.3.2.8 void Controller::RunVRP ( )

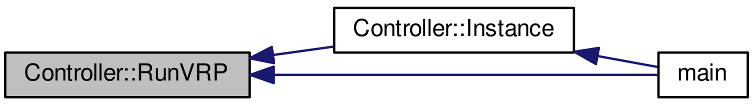
Runs all the main functions

This function sets and call the tabu search and optimal functions. If the routines do not improves the solutions set stop.

Here is the call graph for this function:



Here is the caller graph for this function:

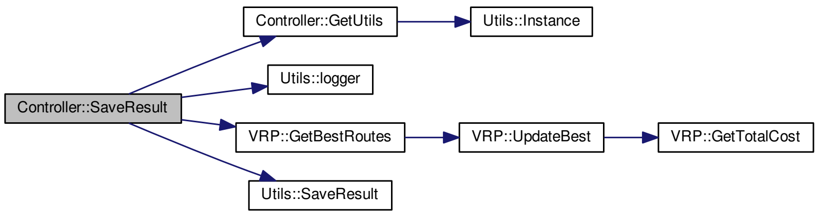


3.3.2.9 void Controller::SaveResult ( )

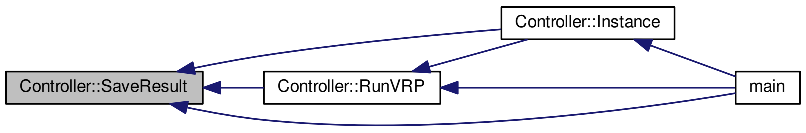
Save results.

The results from the program are saved into a JSON file called 'output.json' in the same folder of the executable file.

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3 Member Data Documentation

3.3.3.1 int Controller::finalCost `[private]`

3.3.3.2 int Controller::initCost `[private]`

3.3.3.3 int Controller::MAX_TIME_MIN `[private]`

3.3.3.4 std::chrono::high_resolution_clock::time_point Controller::startTime `[private]`

3.3.3.5 VRP∗ Controller::vrp `[private]`

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

- actor/Controller.h
- actor/Controller.cpp

3.4 Customer Class Reference

`#include <Customer.h>`

Public Member Functions

- Customer & operator= (const Customer &c)

  Overriding of the "=" operator to assign a customer to another
- bool operator== (const Customer &c) const

  Overriding '==' operator to evaluate two customers
- bool operator!= (const Customer &c) const

  Overriding '!=' operator to evaluate two customers
- Customer ()
- Customer (std::string n, int x, int y, int r, int t)

  Constructor.
- Customer (std::string n, int x, int y)

  Constructor.
- ∼Customer ()

  Destructor.

Public Attributes

- std::string name
- int x
- int y
- int request
- int serviceTime

Friends

- bool operator< (const Customer &c1, const Customer &c2)

  Overriding '<' operator to evaluate two customers
- std::ostream & operator<< (std::ostream &out, Customer c)

  Overriding '<<' operator for printing the customer

3.4.1 Constructor & Destructor Documentation

3.4.1.1 Customer::Customer ( ) [inline]

3.4.1.2 Customer::Customer (std::string *n*, int *x*, int *y*, int *r*, int *t*) [inline]

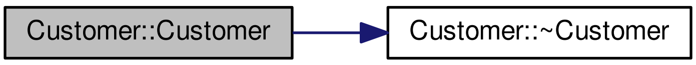
Constructor.

Constructor

3.4.1.3 Customer::Customer (std::string *n*, int *x*, int *y*) [inline]

Constructor.

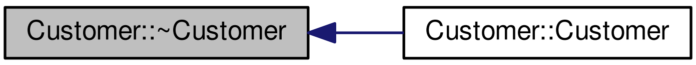
Here is the call graph for this function:



3.4.1.4 Customer::∼Customer ( )

Destructor.

Here is the caller graph for this function:



3.4.2 Member Function Documentation

3.4.2.1 bool Customer::operator!= (const Customer & *c*) const `[inline]`

Overriding '!=' operator to evaluate two customers

3.4.2.2 Customer& Customer::operator= (const Customer & *c*) `[inline]`

Overriding of the "=" operator to assign a customer to another

3.4.2.3 bool Customer::operator== (const Customer & *c*) const `[inline]`

Overriding '==' operator to evaluate two customers

3.4.3 Friends And Related Function Documentation

3.4.3.1 bool operator< (const Customer & *c1,* const Customer & *c2*) `[friend]`

Overriding '<' operator to evaluate two customers

3.4.3.2 std::ostream& operator<< (std::ostream & *out,* Customer *c*) `[friend]`

Overriding '<<' operator for printing the customer

3.4.4 Member Data Documentation

3.4.4.1 std::string Customer::name

Name of the customer

3.4.4.2 int Customer::request

Quantity request from the customer

3.4.4.3 int Customer::serviceTime

Time for serving the customer

3.4.4.4 int Customer::x

Coordinate X of the customer

3.4.4.5 int Customer::y

Coordinate Y of the customer

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

- actor/Customer.h
- actor/Customer.cpp

3.5 Edge Class Reference

```
#include <Edge.h>
```

Classes

- class ConstructionToken

Public Member Functions

- Edge (const Edge &)
- Edge (const ConstructionToken &, int)

  constructor

Public Attributes

- int weight

3.5.1 Constructor & Destructor Documentation

3.5.1.1 Edge::Edge (const Edge &) [default]

3.5.1.2 Edge::Edge (const ConstructionToken & , int *w*)

constructor

Constructor of Edge.

Parameters

in	*w*	Weight of the arch

3.5.2 Member Data Documentation

3.5.2.1 int Edge::weight

Weight of the Edge

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

- lib/Edge.h
- lib/Edge.cpp

3.6 Graph Class Reference

```
#include <Graph.h>
```

Public Member Functions

- Graph ()=default
- void InsertVertex (Customer &)

    Insert a vertex.
- void InsertEdge (Customer &, Customer &, int)

    Insert an Edge.
- void RemoveEdge (Customer &, Customer &)

    Remove an edge.
- std::multimap< int, Customer > sortV0 ()

    Sort the customers by distance.
- std::multimap< int, Customer > GetNeighborhood (Customer) const

    Find the neighborhood of a customer.
- std::pair< Customer, int > GetCosts (const Customer &, const Customer &) const

    Return the weight of an edge.

Protected Member Functions

- void InsertVertex (Customer &, Vertex &)

    Insert a vertex.

Private Attributes

- std::map< Customer, Vertex > vertexes

3.6.1 Constructor & Destructor Documentation

3.6.1.1 Graph::Graph ( ) [default]

3.6.2 Member Function Documentation

3.6.2.1 std::pair< Customer, int > Graph::GetCosts (const Customer & *from*, const Customer & *to*) const

Return the weight of an edge.

This function compute the cost of travelling from a customer to another.

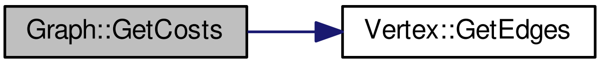
Parameters

in	*from*	The starting customer
in	*to*	The ending customer.

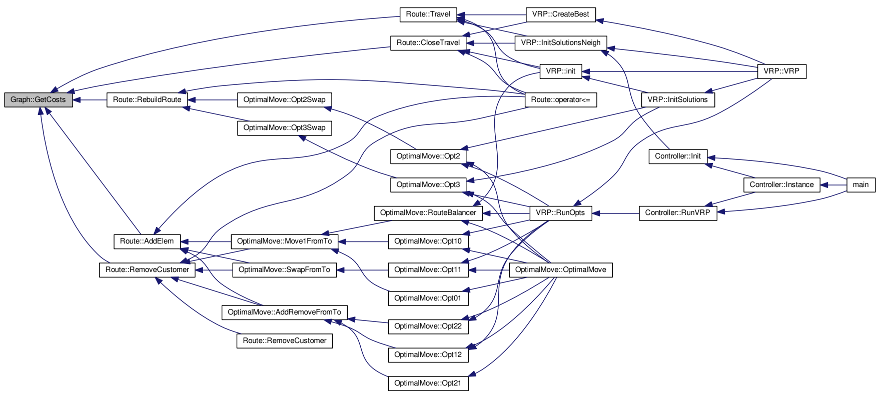
Returns

The cost of the travel

Here is the call graph for this function:



Here is the caller graph for this function:



3.6.2.2 std::multimap< int, **Customer** > Graph::GetNeighborhood (**Customer** *c*) const

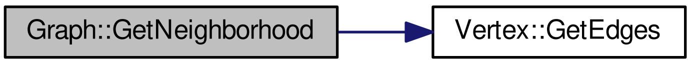
Find the neighborhood of a customer.

This function sorts the neighborhood of a customer by distance; the order is crescent.

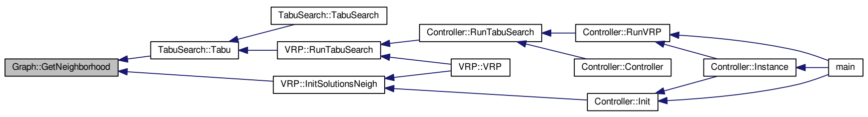
Returns

The map of customer sorted

Here is the call graph for this function:



Here is the caller graph for this function:



3.6.2.3 void Graph::InsertEdge (**Customer** & *node,* **Customer** & *new_edge,* int *weight*)

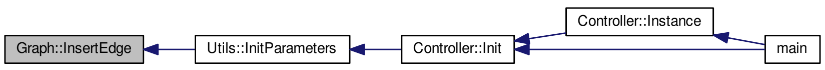
Insert an Edge.

Insert an edge with weight from a customer to another.

Parameters

in	*node*	The starting customer
in	*new_edge*	The destination customer
in	*weight*	The weight of the edge

Here is the caller graph for this function:



3.6.2.4 void Graph::InsertVertex (**Customer** & *cust*)

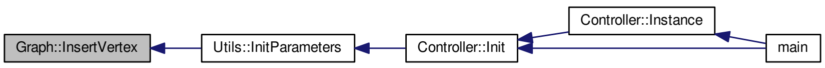
Insert a vertex.

Create and insert a vertex in the graph.

Parameters

`in`	*cust*	The customer who form the vertex

Here is the caller graph for this function:



3.6.2.5 void Graph::InsertVertex (**Customer** & *c*, **Vertex** & *v*) `[protected]`

Insert a vertex.

Insert a vertex in the graph.

Parameters

`in`	*c*	The customer who form the vertex
`in`	*v*	The vertex created

3.6.2.6 void Graph::RemoveEdge (**Customer** & *node*, **Customer** & *edge*)

Remove an edge.

Parameters

`in`	*node*	The customer which the edge start
`in`	*edge*	The customer which the edge finish

3.6.2.7 std::multimap< int, **Customer** > Graph::sortV0 ( )

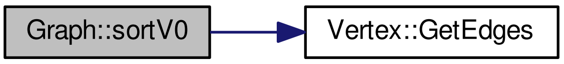
Sort the customers by distance.

This function sorts the customer by distance from the depot; the order is crescent.

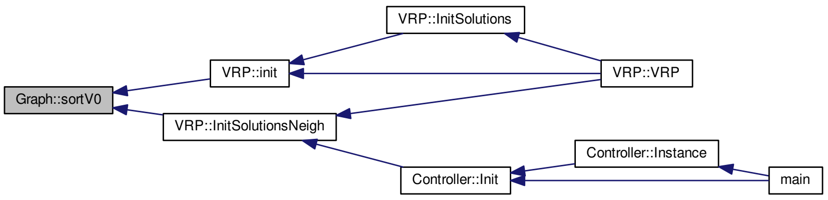
Returns

The map of customer sorted

Here is the call graph for this function:



Here is the caller graph for this function:



3.6.3 Member Data Documentation

3.6.3.1 std::map< Customer, Vertex > Graph::vertexes [private]

Map of vertexes: for each customer the vertex in input or output

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

- lib/Graph.h
- lib/Graph.cpp

3.7 OptimalMove Class Reference

`#include <OptimalMove.h>`

Public Member Functions

- OptimalMove ()
- void CleanVoid (Routes &)

Remove all void routes.
- int Opt10 (Routes &, bool)

  Move a customer from a route to another.
- int Opt01 (Routes &, bool)

  Move a customer from a route to another.
- int Opt11 (Routes &, bool)

  Swap two customers from routes.
- int Opt21 (Routes &, bool)

  This function combine Opt01 and Opt11.
- int Opt12 (Routes &, bool)

  This function combine Opt01 and Opt11.
- int Opt22 (Routes &, bool)

  This function combines Opt01 and Opt11.
- void RouteBalancer (Routes &)

  Try to balance the routes.
- bool Opt2 (Routes &)

  Reorder the customers of route to delete cross over path.
- bool Opt3 (Routes &)

  Reorder the customers of route to delete cross over path.

Private Member Functions

- Route Opt2Swap (Route, Customer, Customer)

  Swap two customer in a route
- Route Opt3Swap (Route, Customer, Customer, Customer, Customer)

  Swap three customer in a route
- float UpdateDistanceAverage (Routes)
- bool Move1FromTo (Route &, Route &, bool)

  Move a customer from a route to another.
- bool SwapFromTo (Route &, Route &, bool)

  Swap two random customer from two routes.
- bool AddRemoveFromTo (Route &, Route &, int, int, bool)

  Move some customers from the routes.

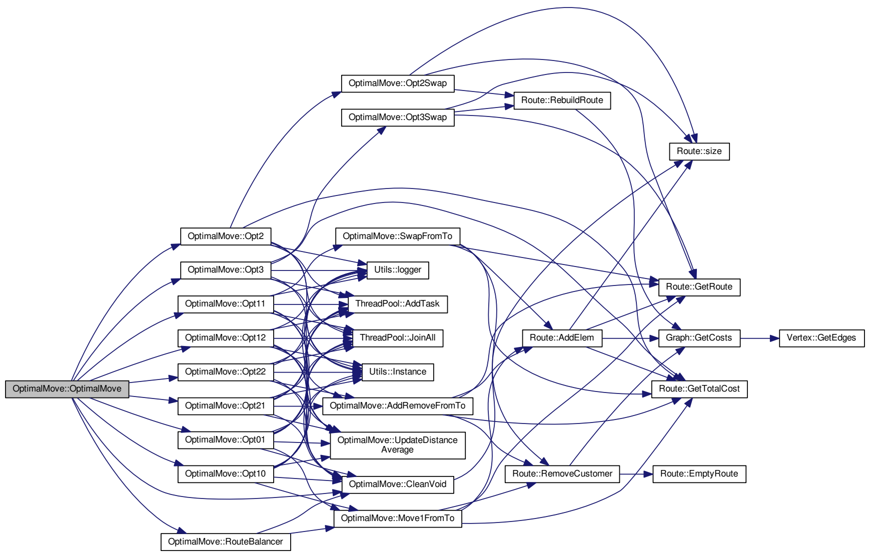
Private Attributes

- std::mutex mtx
- const unsigned cores

3.7.1 Constructor & Destructor Documentation

3.7.1.1 OptimalMove::OptimalMove ( ) `[inline]`

Here is the call graph for this function:



3.7.2 Member Function Documentation

3.7.2.1 bool OptimalMove::AddRemoveFromTo (Route & *source*, Route & *dest*, int *nInsert*, int *nRemove*, bool *force*) `[private]`

Move some customers from the routes.

This function moves a number of customers from two routes and find the best move checking all possible route configurations.

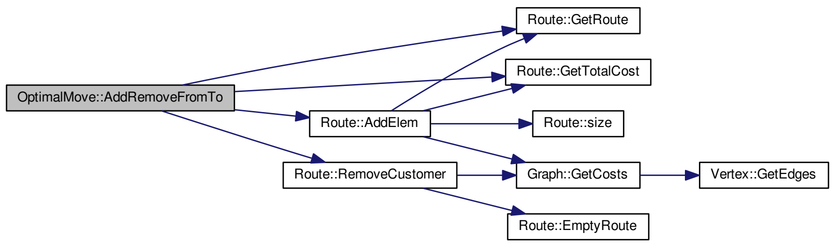
Parameters

in	*source*	Route where to choose a random customer
in	*dest*	Route destination
in	*nInsert*	Number of customer to move from 'source' to 'dest'
in	*nRemove*	Number of customer to move from 'dest' to 'source'
in	*force*	If needs to find the worst combination

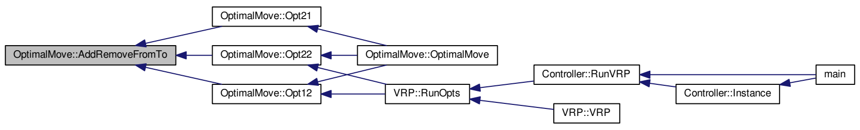
Returns

True if the customer is moved.

Here is the call graph for this function:



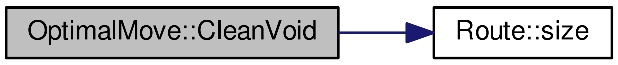
Here is the caller graph for this function:



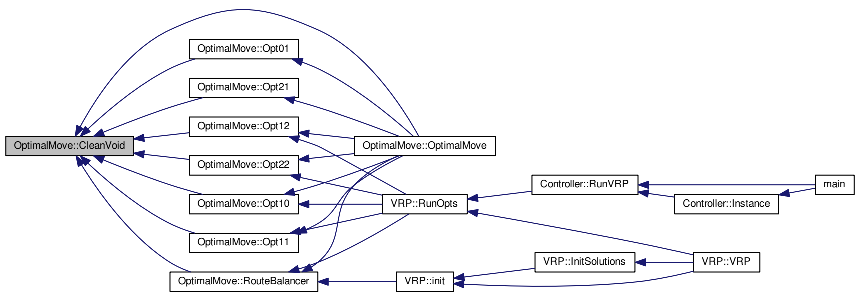
3.7.2.2 void OptimalMove::CleanVoid (**Routes** & *routes*)

Remove all void routes.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.3 bool OptimalMove::Move1FromTo (Route & *source*, Route & *dest*, bool *force*) `[private]`

Move a customer from a route to another.

This function tries to move a customer from a route to another trying in every possible position if and only if the movement results in an improvement of the total cost in both routes.

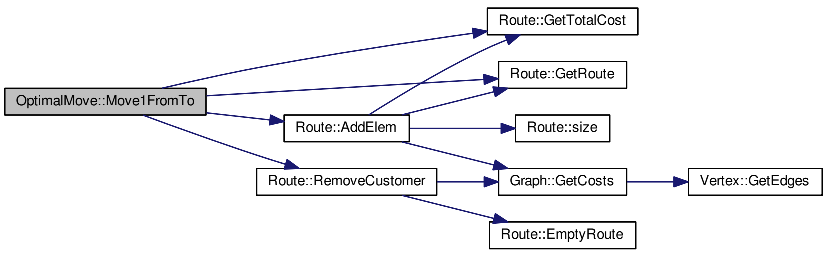
Parameters

`in`	*source*	Route where to choose a random customer
`in`	*dest*	Route destination
`in`	*force*	Force the movement

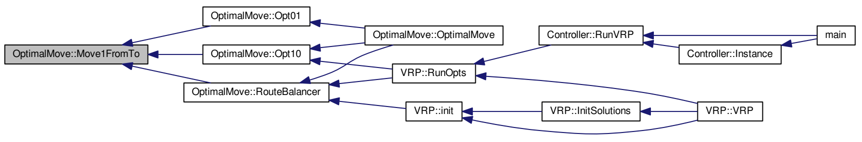
Returns

True if the customer is moved.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.4 int OptimalMove::Opt01 (Routes & *routes,* bool *force*)

Move a customer from a route to another.

This opt function try to move, for every route, a customer from a route to another and remove empty route.

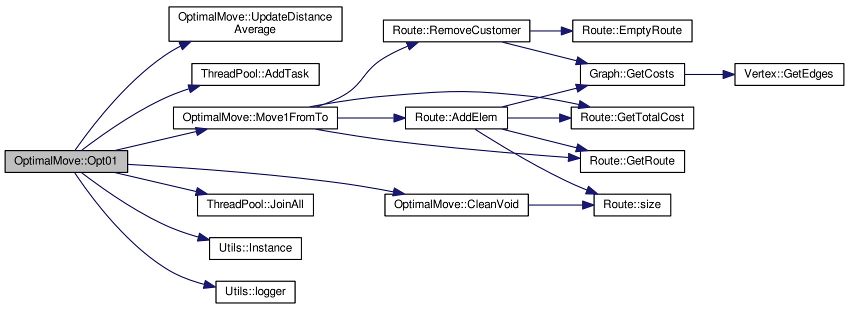
Parameters

in	*routes*	The routes to edit
in	*force*	If needs to find the worst combination

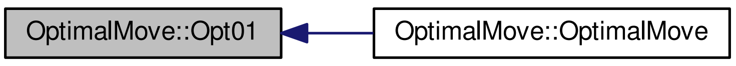
Returns

True if the routes are improves

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.5 int OptimalMove::Opt10 (Routes & *routes*, bool *force*)

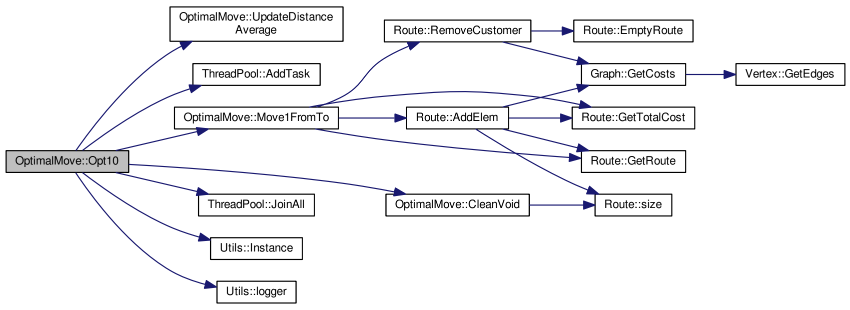
Move a customer from a route to another.

This opt function try to move, for every route, a customer from a route to another and removes empty route.

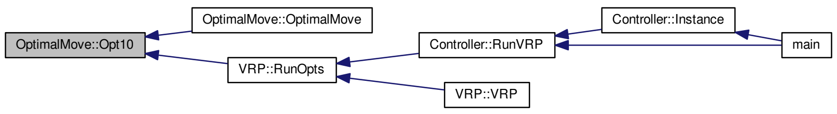
Returns

True if the routes are improved

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.6 int OptimalMove::Opt11 (Routes & *routes*, bool *force*)

Swap two customers from routes.

This opt function try to swap, for every route, a random customer from a route with another random customer from the next.

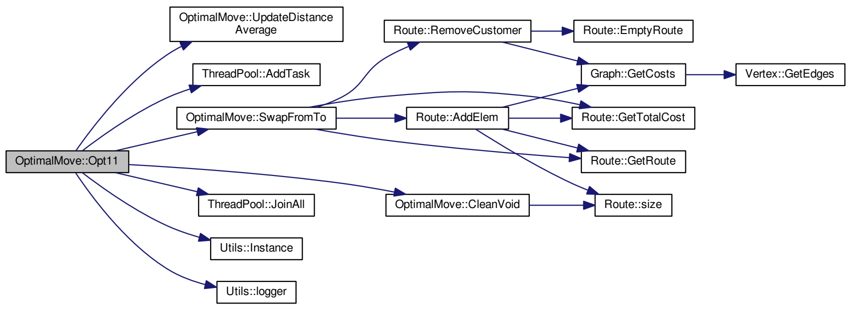
Parameters

in	*routes*	The routes to edit
in	*force*	If needs to find the worst combination

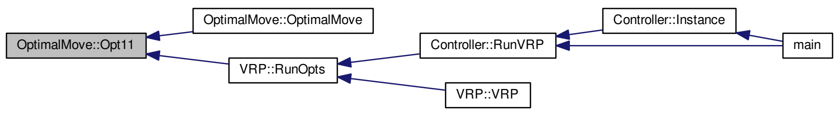
Returns

True if the routes are improves

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.7 int OptimalMove::Opt12 (Routes & *routes*, bool *force*)

This function combine Opt01 and Opt11.

This function swap two customers from the routes and moves one customer from the second to the first route.

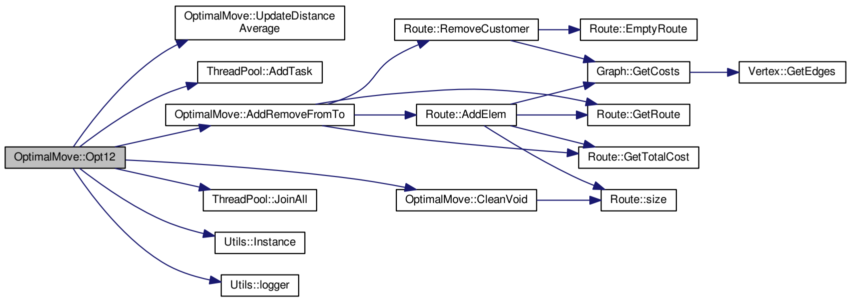
Parameters

`in`	*routes*	The routes to edit
`in`	*force*	If needs to find the worst combination

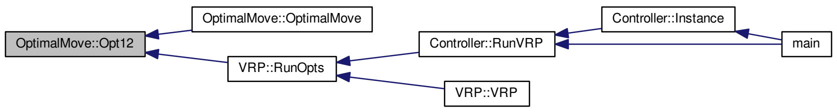
Returns

True if the routes are improves

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.8 bool OptimalMove::Opt2 (Routes & *routes*)

Reorder the customers of route to delete cross over path.

For every route swap two customer with distance lower than the average distance of the route and save the best solution.

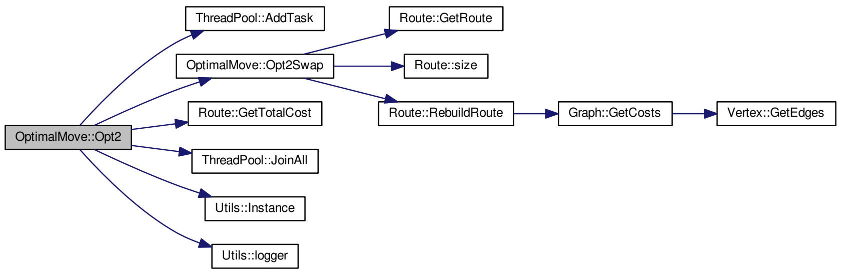
Parameters

`in`	*routes*	The routes to edit

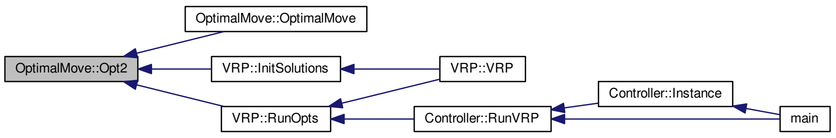
Returns

True if routes are improved

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.9 int OptimalMove::Opt21 (Routes & *routes,* bool *force*)

This function combine Opt01 and Opt11.

This function swap one customers from each routes and moves one customer from the first to the second.

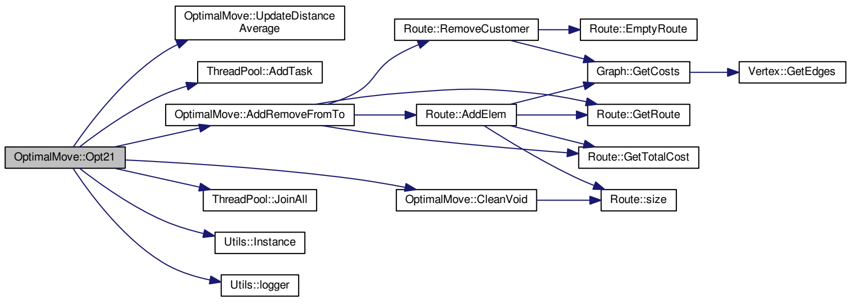
Parameters

in	*routes*	The routes to edit
in	*force*	If needs to find the worst combination

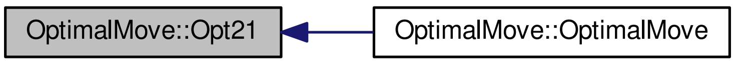
Returns

True if the routes are improves

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.10 int OptimalMove::Opt22 (Routes & *routes,* bool *force*)

This function combines Opt01 and Opt11.

This function swaps two customers from the first route with two customers from the second.

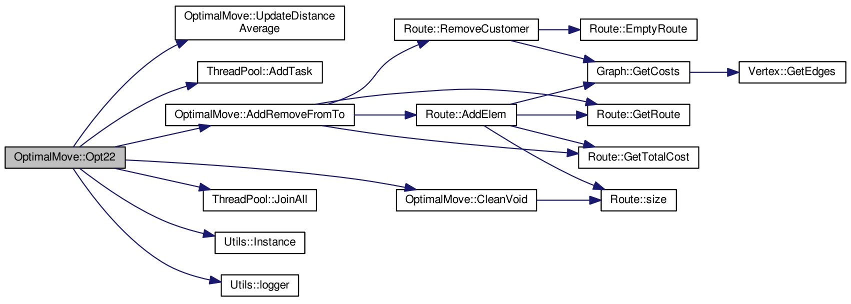
Parameters

`in`	*routes*	The routes to edit
`in`	*force*	If needs to find the worst combination

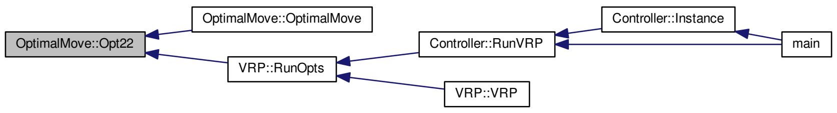
Returns

True if the routes are improved

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.11 Route OptimalMove::Opt2Swap (Route *route,* Customer *i,* Customer *k*) `[private]`

Swap two customer in a route

This function swap two customers in a route.

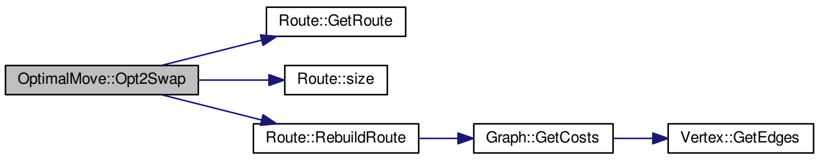
Parameters

`in`	*route*	The route to work with
`in`	*i*	The first customer to swap
`in`	*k*	The second customer to swap

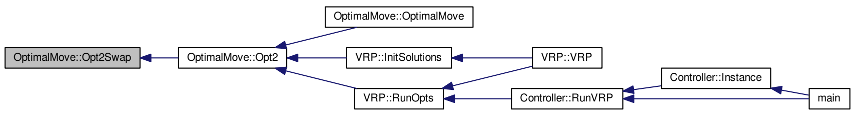
Returns

The new route with customers swapped

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.12 bool OptimalMove::Opt3 (Routes & *routes*)

Reorder the customers of route to delete cross over path.

For every route swap two customer with distance lower than the average distance (the route crosses over itself) of the route and save the best solution.

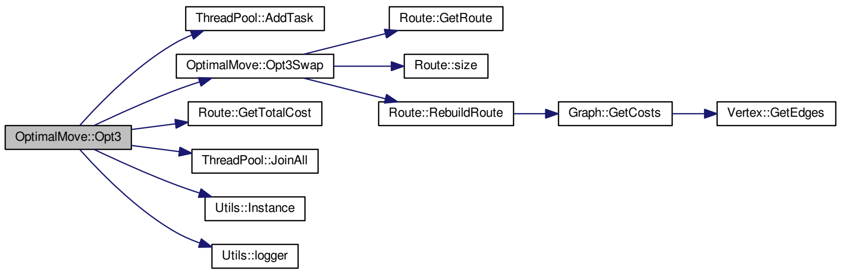
Parameters

`in`	*routes*	The routes to edit

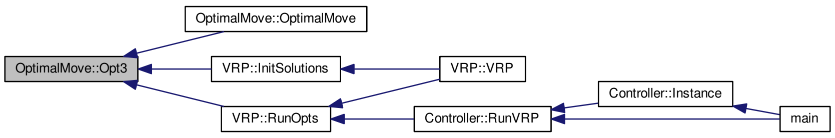
Returns

True if routes are improved

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.13 Route OptimalMove::Opt3Swap (Route *route*, Customer *i*, Customer *k*, Customer *l*, Customer *m*) `[private]`

Swap three customer in a route

This function swap two customers in a route.

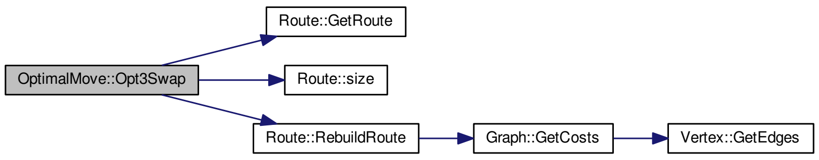
Parameters

`in`	*route*	The route to work with
`in`	*i*	The first customer to swap
`in`	*k*	The second customer to swap
`in`	*l*	The third customer to swap
`in`	*m*	The fourth customer to swap

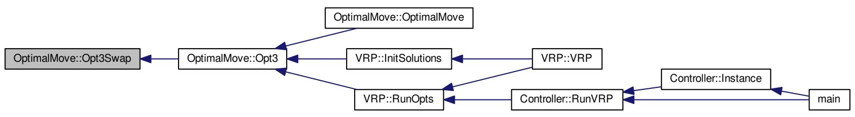
Returns

The new route with customers swapped

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.14 void OptimalMove::RouteBalancer (**Routes** & *routes*)

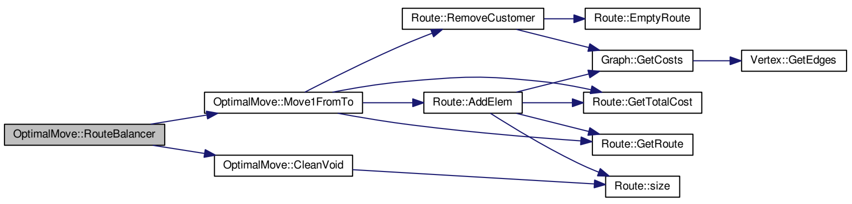
Try to balance the routes.

Find route with one or two customers and move it/them to another route: execute an Opt10 to balance the route and get more occupancy.

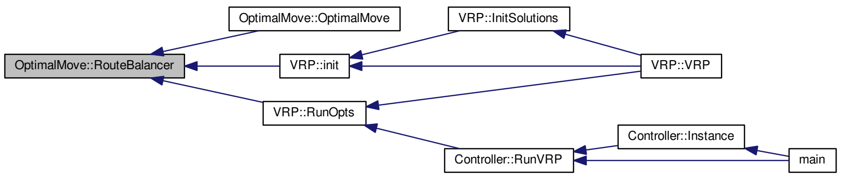
Parameters

in	*routes*	The routes to edit

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.2.15 bool OptimalMove::SwapFromTo (Route & *source*, Route & *dest*, bool *force*) [private]

Swap two random customer from two routes.

This function swap two customers from two routes.

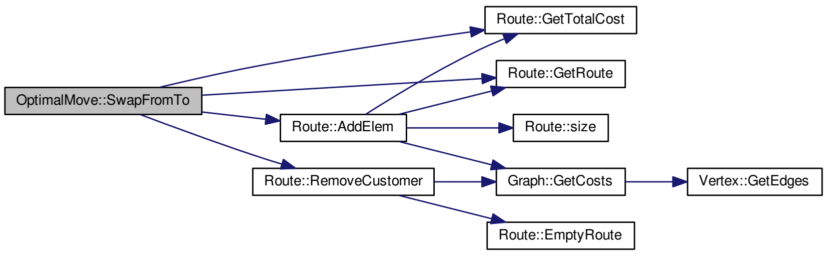
Parameters

in	*source*	First route
in	*dest*	Second route
in	*force*	If needs to find the worst combination

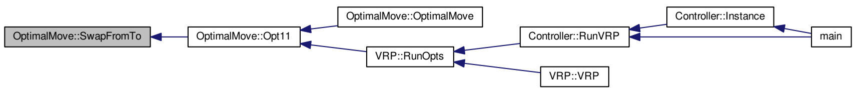
Returns

True is the swap is successful

Here is the call graph for this function:

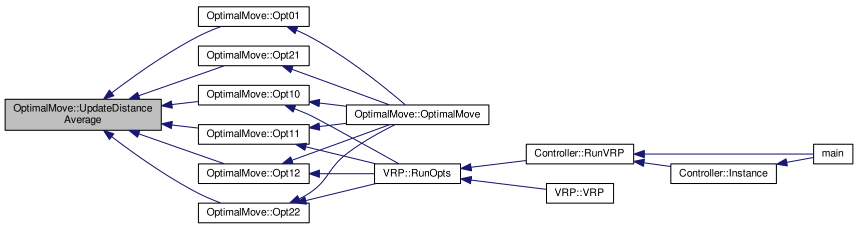


Here is the caller graph for this function:



3.7.2.16 float OptimalMove::UpdateDistanceAverage (**Routes** *routes*) [private]

Here is the caller graph for this function:



3.7.3 Member Data Documentation

3.7.3.1 const unsigned OptimalMove::cores [private]

3.7.3.2 std::mutex OptimalMove::mtx [private]

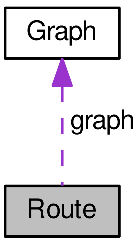
The documentation for this class was generated from the following files:

- lib/OptimalMove.h
- lib/OptimalMove.cpp

3.8 Route Class Reference

```
#include <Route.h>
```

Collaboration diagram for Route:



Public Member Functions

- Route & operator= (const Route &r)

Overriding of the "=" operator for assign a route to another

- bool operator!= (const Route &r) const

  Overriding '!=' operator to evaluate two routes
- bool operator== (const Route &r) const

  Overriding '==' operator to evaluate two routes
- bool operator< (const Route &r) const

  Overriding '<' operator to evaluate two routes
- bool operator<= (const Route &r) const

  Overriding '>=' operator to evaluate two routes
- Route (const int, const float, const Graph, const float, const float)

  constructor
- void CloseTravel (const Customer)

  Close a route.
- bool CloseTravel (const Customer, const Customer)

  Close a route with the last customer.
- void PrintRoute () const

  Print this route.
- bool Travel (const Customer, const Customer)

  Travel from one customer to another
- int size () const

  Return the size (number of customers) of a route.
- std::list< StepType > * GetRoute ()

  Get the pointer to the route list.
- bool AddElem (const Customer)

  Add a customer to this route.
- bool AddElem (const std::list< Customer > &)

  Add a customer to this route.
- void RemoveCustomer (std::list< StepType >::iterator &)

  Remove a customer from a route.
- bool RemoveCustomer (const Customer)

  Remove a customer.
- int GetTotalCost () const

  Return the cost of the route
- void SetAverageCost ()

  Compute the average cost of all paths
- float GetAverageCost () const

  Get the average cost of all paths
- void GetUnderAverageCustomers (std::list< Customer > &)

  List all customers with cost path lower the average.
- float GetDistanceFrom (Route)

  Get the distance from two routes
- bool FindCustomer (const Customer)

  Find a customer in the route
- bool RebuildRoute (std::list< Customer >)

  Rebuild the route starting from a list of customers
- float Evaluate ()

  Quality assessment of the route.

Protected Member Functions

- void EmptyRoute (const Customer)

  Clear a route.

Protected Attributes

- std::list< StepType > route

Private Attributes

- int initialCapacity
- float initialWorkTime
- int capacity
- float workTime
- int totalCost
- float averageCost
- float TRAVEL_COST
- float ALPHA
- Graph graph

Friends

- std::ostream & operator<< (std::ostream &out, const Route &r)

  Overriding of the "<<" operator to print the route

3.8.1 Constructor & Destructor Documentation

3.8.1.1 Route::Route (const int *c*, const float *wt*, const **Graph** *g*, const float *costTravel*, const float *alphaParam*)

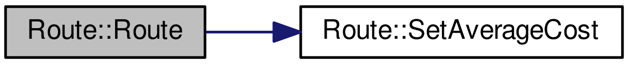
constructor

Constructor of **Route**.

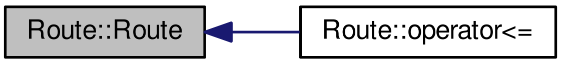
Parameters

in	*c*	Initial capacity of the vehicle.
in	*wt*	Initial work time of the driver.
in	*g*	Graph of the customers.
in	*costTravel*	Cost parameter for each travel.
in	*alphaParam*	Alpha parameter for router evalutation.

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.2 Member Function Documentation

3.8.2.1 bool Route::AddElem (const Customer *c*)

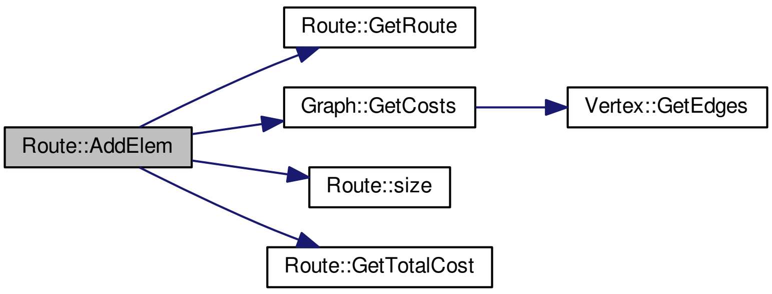
Add a customer to this route.

This function add a customer in the best position of a route respecting the constraints: add the customer in each possible position, then execute the best insertion.

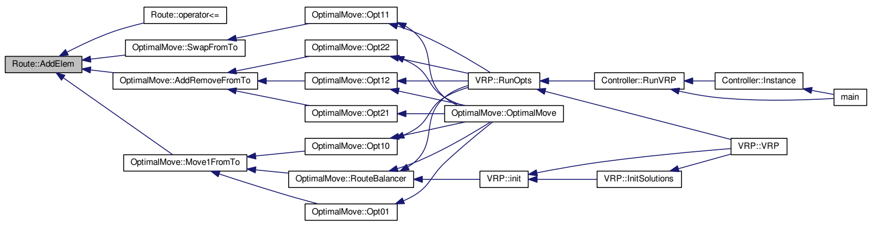
Parameters

`in`	*c*	The customer to insert

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.2.2 bool Route::AddElem (const std::list< **Customer** > & *custs*)

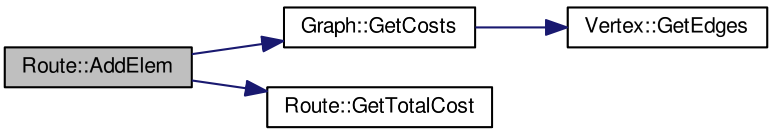
Add a customer to this route.

This function add a list of consecutive customers in the best position of a route respecting the constraints: add the customer in each possible position, then execute the best insertion.

Parameters

`in`	*custs*	List of customers to insert

Here is the call graph for this function:



3.8.2.3 void Route::CloseTravel (const Customer *c*)

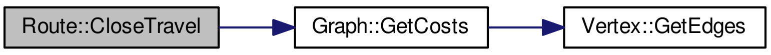
Close a route.

Whenever the capacity or the work time are inadequate close the route: return to depot.

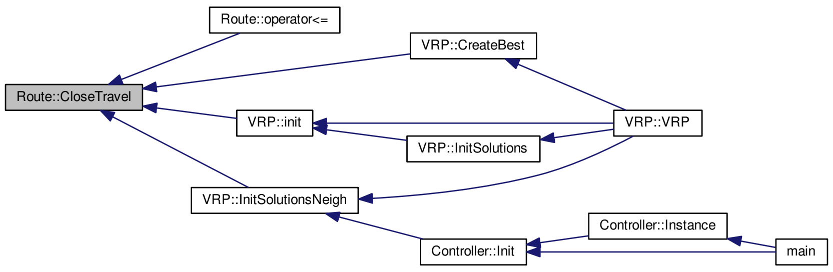
Parameters

`in`	*c*	Last customer to visit

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.2.4 bool Route::CloseTravel (const **Customer** *from,* const **Customer** *depot*)

Close a route with the last customer.

When remaining only one customer to visit, visit it then return to depot.

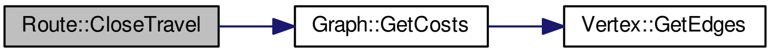
Parameters

`in`	*from*	The last customer to visit
`in`	*depot*	The depot

Returns

True if the customer is visitable

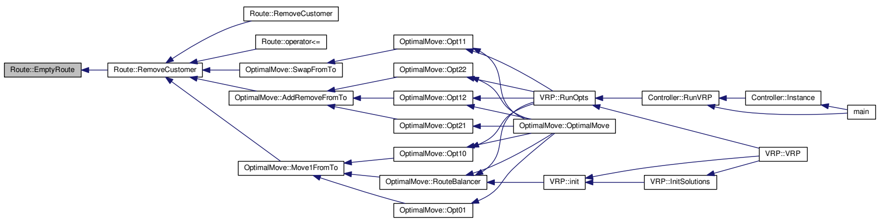
Here is the call graph for this function:



3.8.2.5 void Route::EmptyRoute (const **Customer** *depot*) `[protected]`

Clear a route.

Here is the caller graph for this function:



3.8.2.6 float Route::Evaluate ( )

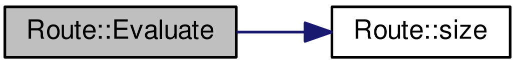
Quality assessment of the route.

This function evaluate the 'quality' of the route checking the occupancy in capacity and time terms. Less is better.

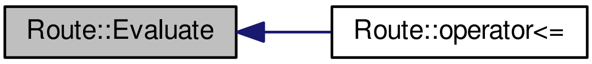
Returns

The value of the assessment.

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.2.7 bool Route::FindCustomer (const **Customer** *c*)

Find a customer in the route

Search for a customer in the route, if it is present return True, otherwise False.

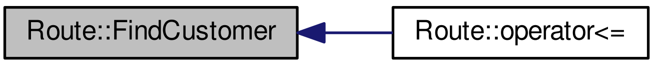
Parameters

`in`	*c*	The customer to search for

Returns

The result

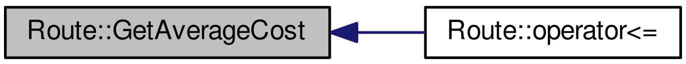
Here is the caller graph for this function:



3.8.2.8 float Route::GetAverageCost ( ) const

Get the average cost of all paths

Here is the caller graph for this function:



3.8.2.9 float Route::GetDistanceFrom (Route *r*)

Get the distance from two routes

The distance from two routes is defined as the minimum distance from each customers of the routes.

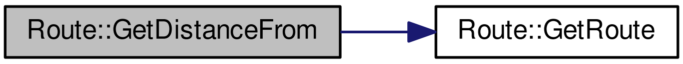
Parameters

`in`	*r*	The route to compare with

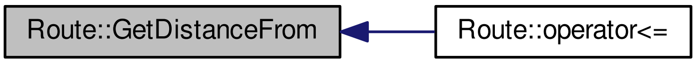
Returns

The distance from the two routes.

Here is the call graph for this function:



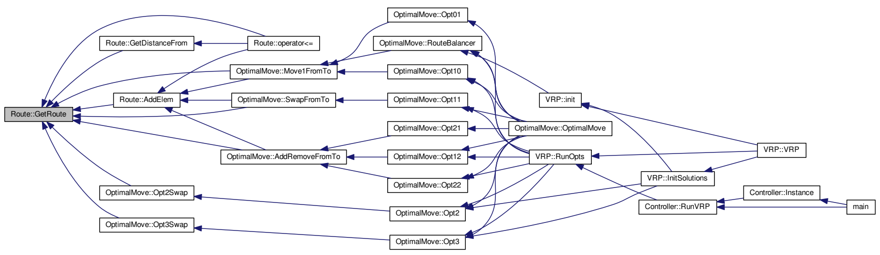
Here is the caller graph for this function:



3.8.2.10 std::list< StepType > ∗ Route::GetRoute ( )

Get the pointer to the route list.

Here is the caller graph for this function:



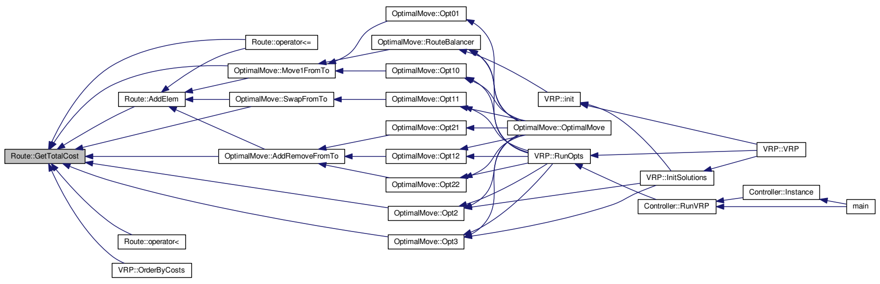
3.8.2.11 int Route::GetTotalCost ( ) const

Return the cost of the route

Returns

Cost of the route

Here is the caller graph for this function:



3.8.2.12 void Route::GetUnderAverageCustomers (std::list< Customer > & *customers*)

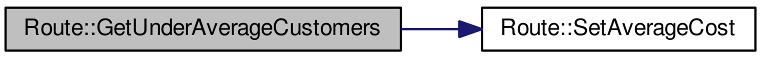
List all customers with cost path lower the average.

Create a list of customers which have a cost path lower than the average cost of the route.

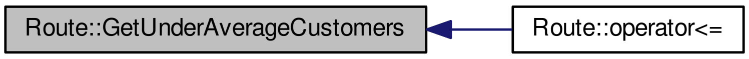
Parameters

in	*customers*	Initial list of customers

Here is the call graph for this function:



Here is the caller graph for this function:



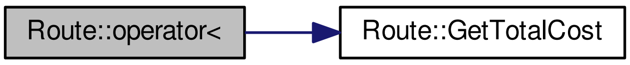
3.8.2.13 bool Route::operator!= (const Route & *r*) const `[inline]`

Overriding '!=' operator to evaluate two routes

3.8.2.14 bool Route::operator< (const Route & *r*) const `[inline]`

Overriding '<' operator to evaluate two routes

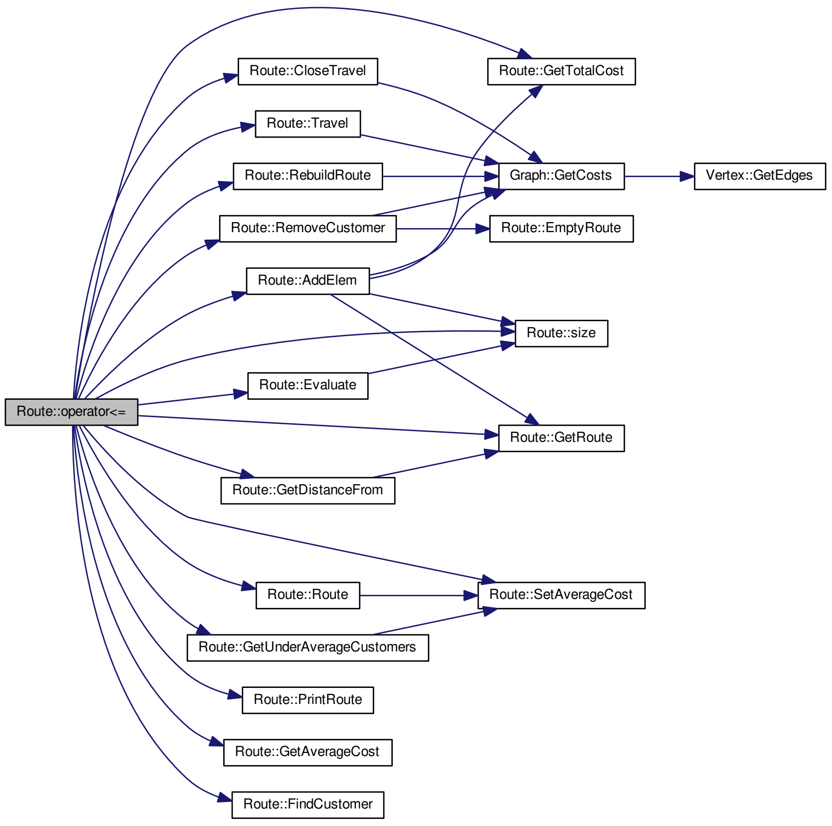
Here is the call graph for this function:



3.8.2.15 bool Route::operator<= (const Route & *r*) const `[inline]`

Overriding '>=' operator to evaluate two routes

Here is the call graph for this function:



3.8.2.16 Route& Route::operator= (const Route & *r*) [inline]

Overriding of the "=" operator for assign a route to another

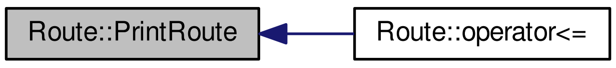
3.8.2.17 bool Route::operator== (const Route & *r*) const [inline]

Overriding '==' operator to evaluate two routes

3.8.2.18 void Route::PrintRoute ( ) const

Print this route.

Here is the caller graph for this function:



3.8.2.19 bool Route::RebuildRoute (std::list< **Customer** > *cust*)

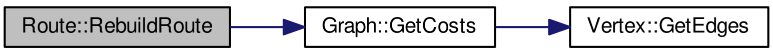
Rebuild the route starting from a list of customers

From a list of customers this function rebuild the route (checking all constraint).

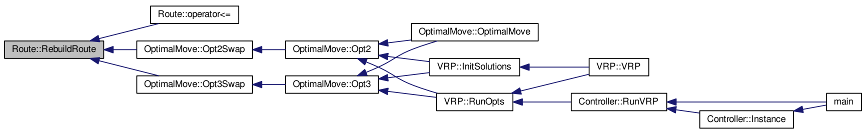
Returns

True if the new route is valid

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.2.20 void Route::RemoveCustomer (std::list< **StepType** >::iterator & *it*)

Remove a customer from a route.

Remove a customer in a position in the route.

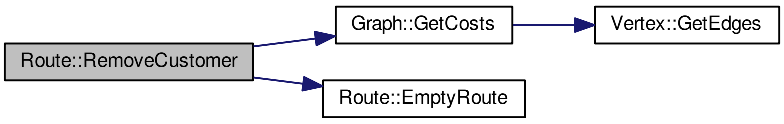
Parameters

`in`	*it*	The position of the customer to remove

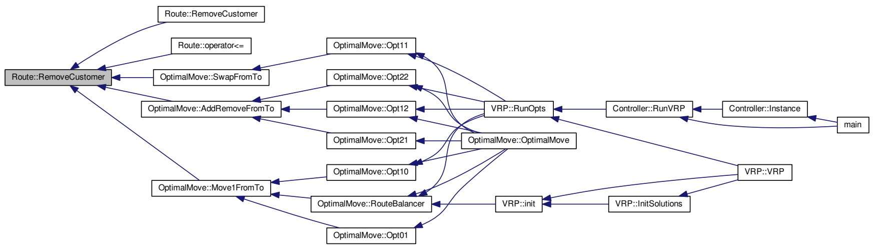
Returns

The state of the operation

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.2.21 bool Route::RemoveCustomer (const Customer *c*)

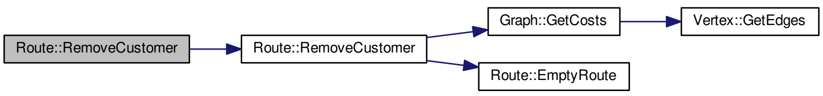
Remove a customer.

Find and remove a customer from the route.

Parameters

in	*c*	The customer to remove

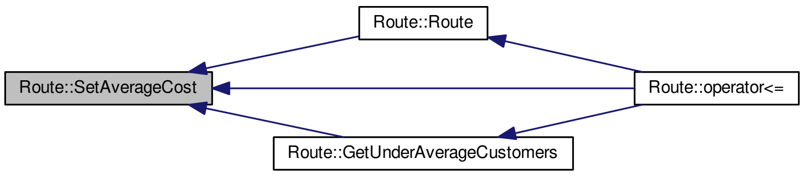
Here is the call graph for this function:



3.8.2.22 void Route::SetAverageCost ( )

Compute the average cost of all paths

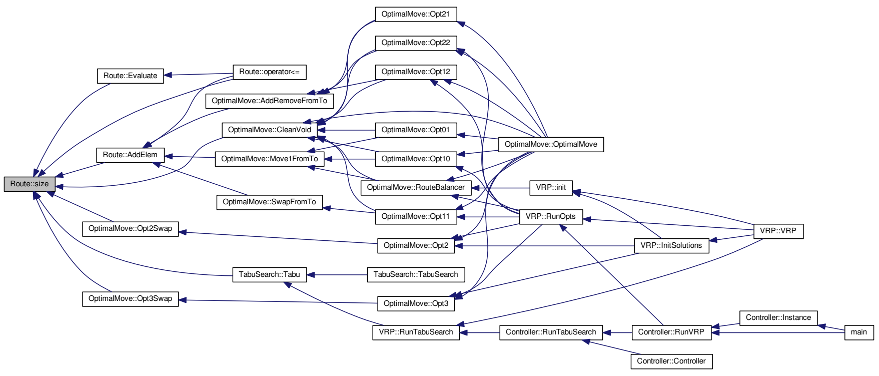
Here is the caller graph for this function:



3.8.2.23 int Route::size ( ) const

Return the size (number of customers) of a route.

Here is the caller graph for this function:



3.8.2.24 bool Route::Travel (const **Customer** *from*, const **Customer** *to*)

Travel from one customer to another

Check if is possibile to travel from a customer to another observing the constraint of capacity and time, if possibile add the travel to the route

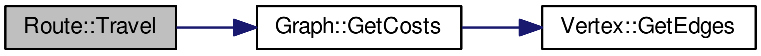
Parameters

`in`	*from*	The source customer
`in`	*to*	The destination customer

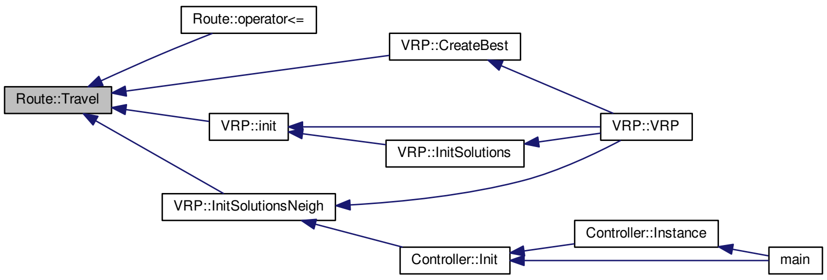
Returns

True if the travel is added to the route

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.3 Friends And Related Function Documentation

3.8.3.1 std::ostream& operator<< (std::ostream & *out,* const Route & *r*) [friend]

Overriding of the "<<" operator to print the route

3.8.4 Member Data Documentation

3.8.4.1 float Route::ALPHA [private]

Alpha parameter for route evaluation

3.8.4.2 float Route::averageCost [private]

Average of all path costs

3.8.4.3 int Route::capacity [private]

Capacity remaining

3.8.4.4 Graph Route::graph [private]

Graph of the customers

3.8.4.5 int Route::initialCapacity [private]

Initial capacity of the route, equals to VRP.capacity

3.8.4.6 float Route::initialWorkTime [private]

Total work time for driver, equals to VRP.workTime

3.8.4.7 std::list< StepType > Route::route [protected]

This list represent the route

3.8.4.8 int Route::totalCost [private]

Total cost of the route: sum of the weight

3.8.4.9 float Route::TRAVEL_COST [private]

Cost parameter for each travel

3.8.4.10 float Route::workTime [private]

Work time remaining

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

- actor/Route.h
- actor/Route.cpp

3.9 TabuList Class Reference

```
#include <TabuList.h>
```

Public Member Functions

- TabuList ()
- TabuList (int n)
- void IncrementSize ()
- void DecrementSize ()
- void AddTabu (Move, int)

Add a tabu move to the list.
- void RemoveTabu (Move)

  Remove a tabu move from the list
- void Clean ()

  Aspiration criteria
- bool Find (Move) const

  Find a move in the list
- float Check (Move) const

  Number of times the move is added to a solution

Private Member Functions

- void FlushTabu ()

  Clear the list

Private Attributes

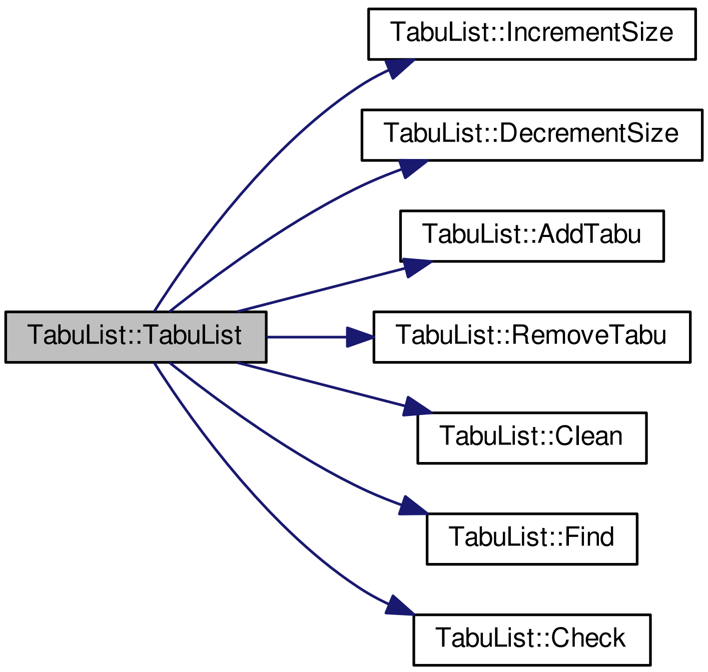
- std::forward_list< TabuElement > tabulist
- std::vector< TabuElement > nonBestMoves
- unsigned size = 7

3.9.1 Constructor & Destructor Documentation

3.9.1.1 TabuList::TabuList ( ) [inline]

3.9.1.2 TabuList::TabuList (int *n*) [inline]

Here is the call graph for this function:



3.9.2 Member Function Documentation

3.9.2.1 void TabuList::AddTabu (Move *m,* int *time*)

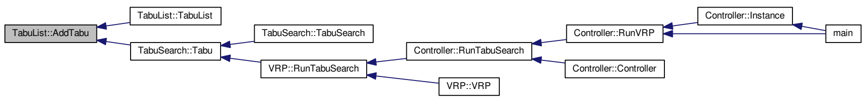
Add a tabu move to the list.

This function insert a pair of Customer and Route and increment the counter of 'how many times was used' the move.

Parameters

in	*m*	The move
in	*time*	Times the move is tabu

Here is the caller graph for this function:



3.9.2.2 float TabuList::Check (Move *m*) const

Number of times the move is added to a solution

Finds and returns the number of times a move is added to a solution

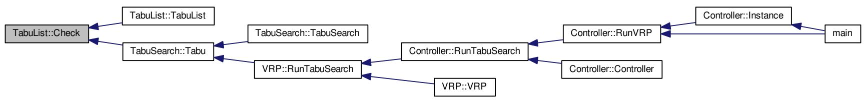
Parameters

in	*m*	The move to search for

Returns

Times the move is added to a solution

Here is the caller graph for this function:

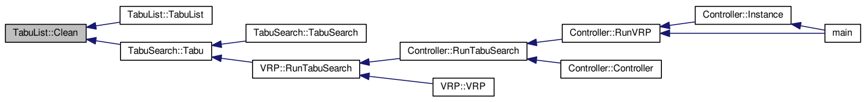


3.9.2.3 void TabuList::Clean ( )

Aspiration criteria

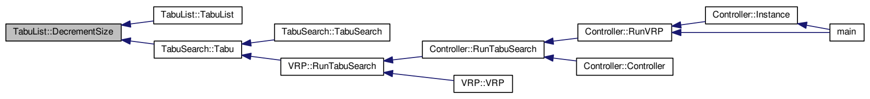
The aspiration criteria decrease the score of tabu moves until their are not tabu anymore.

Here is the caller graph for this function:



3.9.2.4 void TabuList::DecrementSize ( )

Here is the caller graph for this function:



3.9.2.5 bool TabuList::Find (Move *m*) const

Find a move in the list

This function find is a move in the tabulist can be processed; plus, when customer i was previously removed from route k, its reinsertion in that route is forbidden.

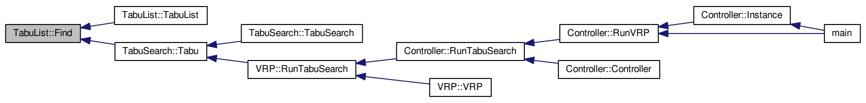
Parameters

in	*m*	The move to find

Returns

If the move is in list

Here is the caller graph for this function:

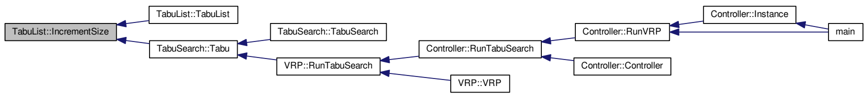


3.9.2.6 void TabuList::FlushTabu ( ) [private]

Clear the list

3.9.2.7 void TabuList::IncrementSize ( )

Here is the caller graph for this function:



3.9.2.8 void TabuList::RemoveTabu (Move *p*)

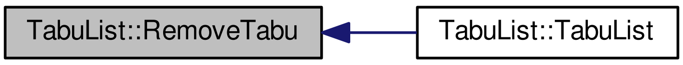
Remove a tabu move from the list

Invalidate all moves which contain the route.

Parameters

in	*p*	The move to remove

Here is the caller graph for this function:



3.9.3 Member Data Documentation

3.9.3.1 std::vector<TabuElement> TabuList::nonBestMoves [private]

3.9.3.2 unsigned TabuList::size = 7 [private]

3.9.3.3 std::forward_list<TabuElement> TabuList::tabulist [private]

List of all tabu moves

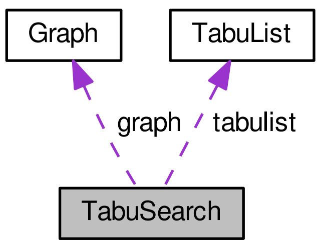
The documentation for this class was generated from the following files:

- actor/TabuList.h
- actor/TabuList.cpp

3.10 TabuSearch Class Reference

```
#include <TabuSearch.h>
```

Collaboration diagram for TabuSearch:



Public Member Functions

- TabuSearch (const Graph &g, const int n)
- void Tabu (Routes &, int)

    Primary function, search for better solution and updates the tabu list

Private Member Functions

- int FindRouteFromCustomer (Customer, Routes)

    Find the route to which a customer belongs
- float Evaluate (Routes)

    Evaluate the assessment of the solution

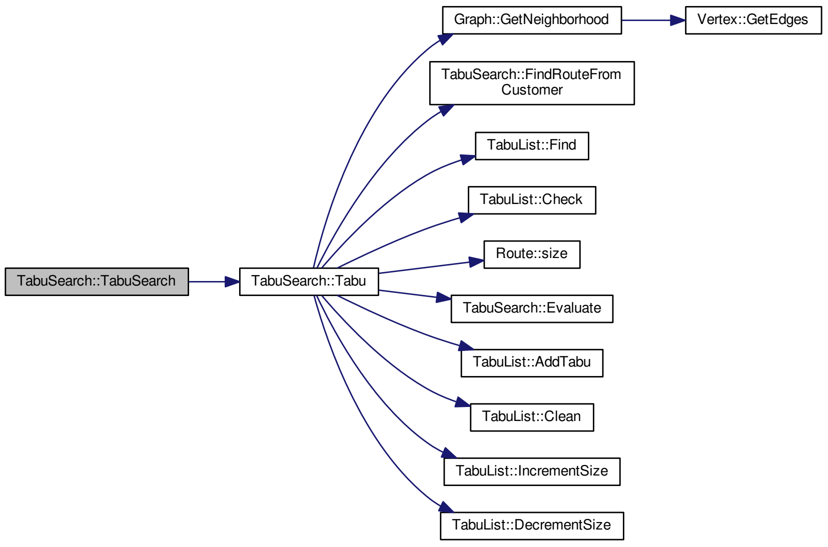
Private Attributes

- Graph graph
- TabuList tabulist
- int numCustomers
- float fitness
- float lambda = 0.0001f

3.10.1 Constructor & Destructor Documentation

3.10.1.1 TabuSearch::TabuSearch (const Graph & *g*, const int *n*) [inline]

Here is the call graph for this function:



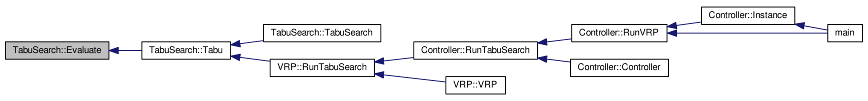
3.10.2 Member Function Documentation

3.10.2.1 float TabuSearch::Evaluate (Routes *l*) [private]

Evaluate the assessment of the solution

This function assess the quality of the solution

Here is the caller graph for this function:



3.10.2.2 int TabuSearch::FindRouteFromCustomer (Customer *c*, Routes *l*) [private]

Find the route to which a customer belongs

This function finds the route to which a customer belongs and return the index of the route.

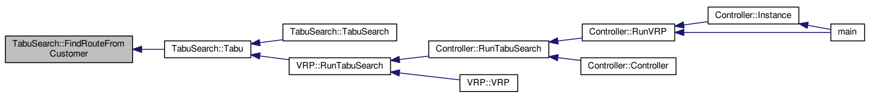
Parameters

in	*c*	The customer to find
in	*l*	The list where find the customer

Returns

The index of the route

Here is the caller graph for this function:

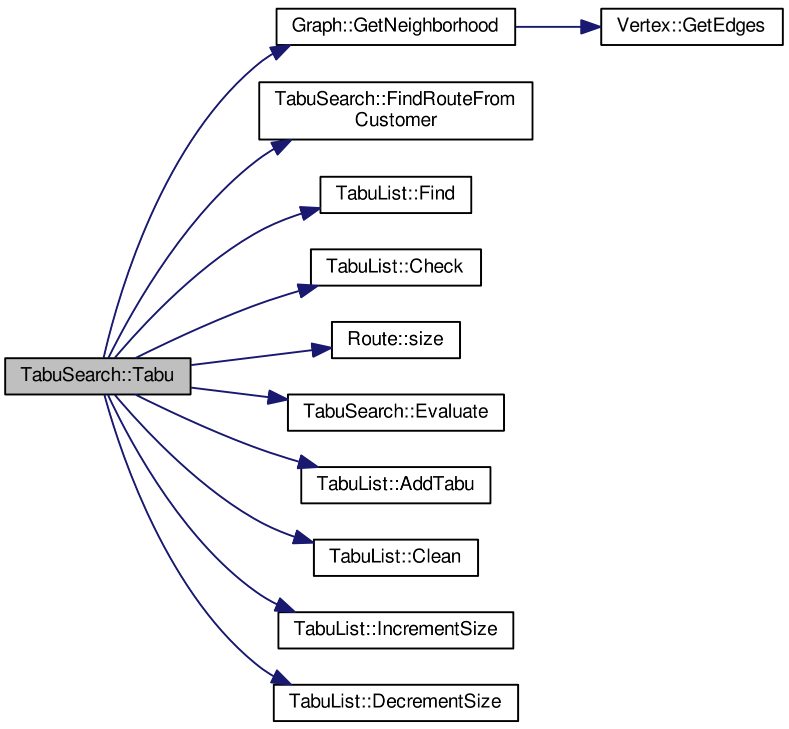


3.10.2.3 void TabuSearch::Tabu (Routes & *routes,* int *times*)

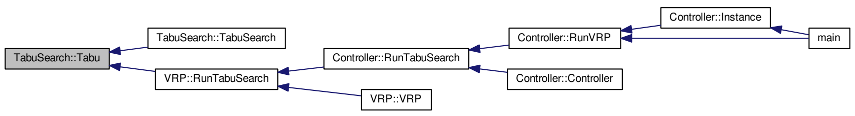
Primary function, search for better solution and updates the tabu list

This function run an iterated local search for a customer and try to insert neighbors customers in the route. If the move isn't legal, updates the tabu list. If the move is tabu but improve then update the route. If no improvement are made choose the best of the worst.

Here is the call graph for this function:



Here is the caller graph for this function:



3.10.3 Member Data Documentation

3.10.3.1 float TabuSearch::fitness [private]

3.10.3.2 Graph TabuSearch::graph [private]

3.10.3.3 float TabuSearch::lambda = 0.0001f [private]

Parameter for penalization of moves

3.10.3.4 int TabuSearch::numCustomers [private]

Number of customers

3.10.3.5 TabuList TabuSearch::tabulist [private]

List of all tabu moves

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

- actor/TabuSearch.h
- actor/TabuSearch.cpp

3.11 ThreadPool Class Reference

```
#include <ThreadPool.h>
```

Public Member Functions

- ThreadPool (int c)
- ∼ThreadPool ()
- void AddTask (std::function< void(void)> job)
- void JoinAll ()

Private Member Functions

- void Run ()

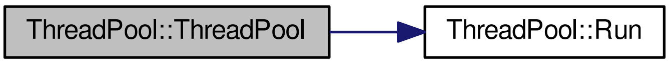
Private Attributes

- unsigned threadCount
- std::vector< std::thread > threads
- std::list< std::function< void(void)> > queue
- std::atomic_bool stop
- std::condition_variable wait_var
- std::mutex queue_mutex

3.11.1 Constructor & Destructor Documentation

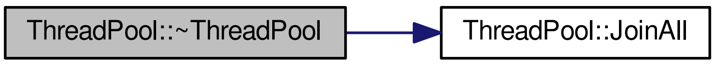
3.11.1.1 ThreadPool::ThreadPool (int *c*) [inline]

Here is the call graph for this function:



3.11.1.2 ThreadPool::∼ThreadPool ( ) [inline]

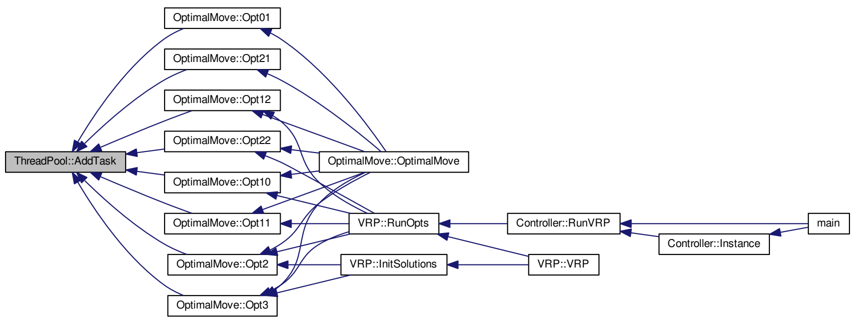
Here is the call graph for this function:



3.11.2 Member Function Documentation

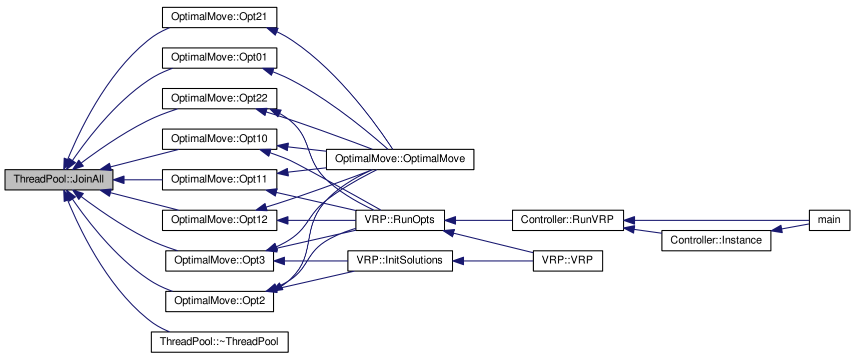
3.11.2.1 void ThreadPool::AddTask (std::function< void(void)> *job*) [inline]

Here is the caller graph for this function:



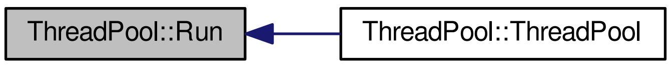
3.11.2.2 void ThreadPool::JoinAll ( ) [inline]

Here is the caller graph for this function:



3.11.2.3 void ThreadPool::Run ( ) [inline], [private]

Here is the caller graph for this function:



3.11.3 Member Data Documentation

3.11.3.1 std::list<std::function<void(void)> > ThreadPool::queue [private]

3.11.3.2 std::mutex ThreadPool::queue_mutex [private]

3.11.3.3 std::atomic_bool ThreadPool::stop [private]

3.11.3.4 unsigned ThreadPool::threadCount [private]

3.11.3.5 std::vector<std::thread> ThreadPool::threads [private]

3.11.3.6 std::condition_variable ThreadPool::wait_var [private]

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

- lib/ThreadPool.h

3.12 Utils Class Reference

```
#include <Utils.h>
```

Public Member Functions

- VRP * InitParameters (int, char **, const float, const float)

  Instantiate all parameters.
- void SaveResult (const std::list< Route >, int)

  Save the result.
- template<typename T >
  void logger (T s, int c=5) const

  Print a log string.

Static Public Member Functions

- static Utils & Instance ()

Public Attributes

- bool verbose = false
- std::string filename = ""

Static Public Attributes

- static const int SUCCESS = 0
- static const int ERROR = 1
- static const int WARNING = 2
- static const int INFO = 3
- static const int VERBOSE = 4

Private Member Functions

- Utils ()
- Utils (Utils const &)=delete
- void operator= (Utils const &)=delete

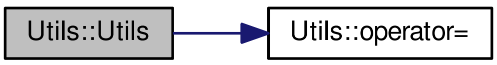
Private Attributes

- rapidjson::Document d
- const char $*$ ANSI_RESET = "\u001B[0m"
- const char $*$ ANSI_RED = "\u001B[1;31m"
- const char $*$ ANSI_GREEN = "\u001B[1;32m"
- const char $*$ ANSI_YELLOW = "\u001B[33m"
- const char $*$ ANSI_BLUE = "\u001B[1;34m"
- const char $*$ ANSI_LIGHTGREEN = "\u001B[32m"
- const char $*$ ANSI_IBLUE = "\e[0;94m"

3.12.1 Constructor & Destructor Documentation

3.12.1.1 Utils::Utils ( ) `[inline]`, `[private]`

Here is the call graph for this function:



3.12.1.2 Utils::Utils (Utils const &) `[private]`, `[delete]`

3.12.2 Member Function Documentation

3.12.2.1 VRP $*$ Utils::InitParameters (int *argc*, char $**$ *argv*, const float *costTravel*, const float *alphaParam*)

Instantiate all parameters.

Parse the input file in JSON format and instantiates all variables for the algorithm.

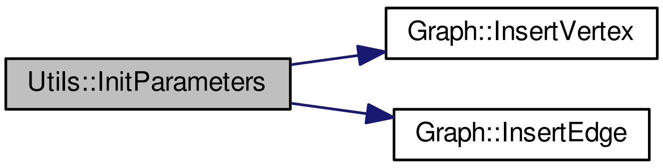
Parameters

`in`	*argc*	Number of arguments passed through command line.
`in`	*argv*	Input file (json).
`in`	*costTravel*	Cost parameter for each travel.
`in`	*alphaParam*	Alpha parameter for router evaluation.

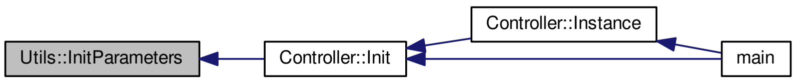
Returns

The pointer to VRP class

Here is the call graph for this function:

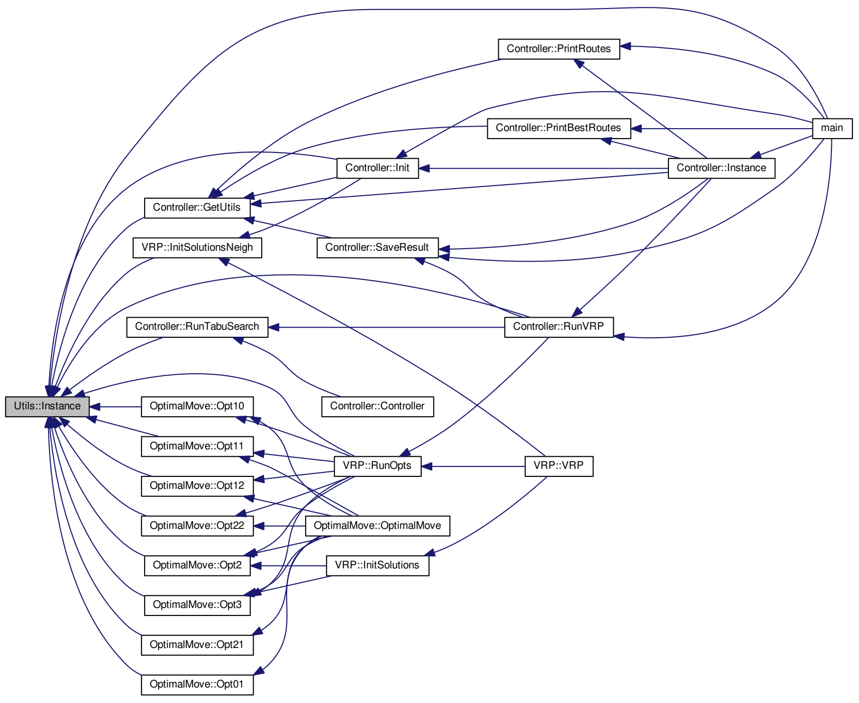


Here is the caller graph for this function:



3.12.2.2 static Utils& Utils::Instance ( ) [inline], [static]

Here is the caller graph for this function:



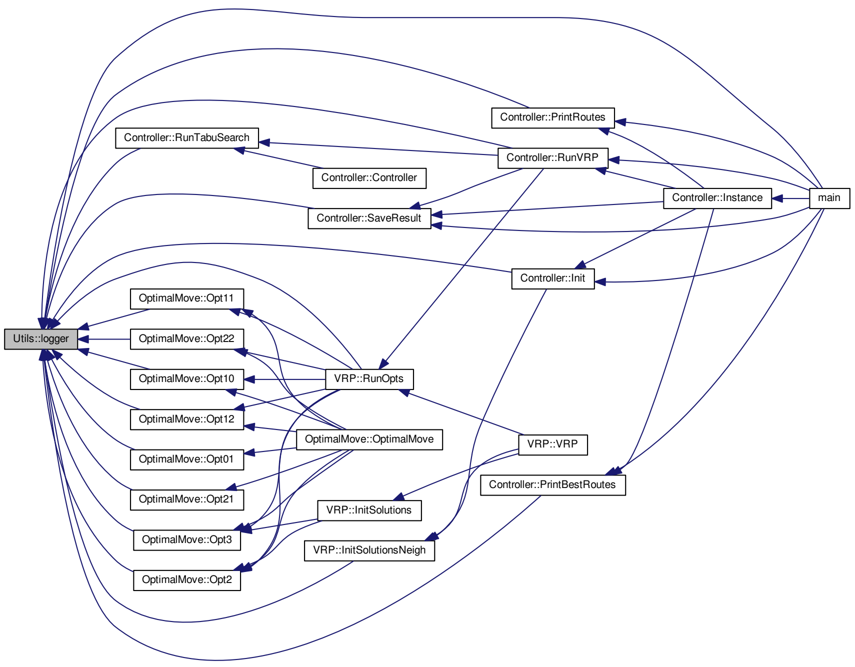
3.12.2.3 template<typename T > void Utils::logger (T *s*, int *c* = 5) const [inline]

Print a log string.

Parameters

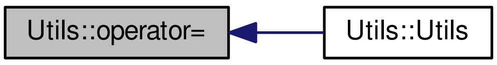
in	*s*	The string to print
in	*c*	The code for log level

Here is the caller graph for this function:



3.12.2.4 void Utils::operator= (Utils const &) [private], [delete]

Here is the caller graph for this function:



3.12.2.5 void Utils::SaveResult (const std::list< Route > *routes,* int *t*)

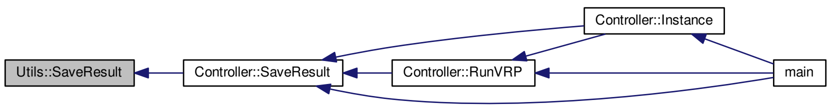
Save the result.

Saves the routes into the input JSON file.

Parameters

in	*routes*	The routes list to save to the file
in	*t*	Execution partial time

Here is the caller graph for this function:



3.12.3 Member Data Documentation

3.12.3.1 const char∗ Utils::ANSI_BLUE = "\u001B[1;34m" [private]

3.12.3.2 const char∗ Utils::ANSI_GREEN = "\u001B[1;32m" [private]

3.12.3.3 const char∗ Utils::ANSI_IBLUE = "\e[0;94m" [private]

3.12.3.4 const char∗ Utils::ANSI_LIGHTGREEN = "\u001B[32m" [private]

3.12.3.5 const char∗ Utils::ANSI_RED = "\u001B[1;31m" [private]

3.12.3.6 const char∗ Utils::ANSI_RESET = "\u001B[0m" [private]

3.12.3.7 const char∗ Utils::ANSI_YELLOW = "\u001B[33m" [private]

3.12.3.8 rapidjson::Document Utils::d [private]

JSON Document

3.12.3.9 const int Utils::ERROR = 1 [static]

Error code

3.12.3.10 std::string Utils::filename = ""

3.12.3.11 const int Utils::INFO = 3 [static]

Simple logging code

3.12.3.12 const int Utils::SUCCESS = 0 [static]

Success code

3.12.3.13 const int Utils::VERBOSE = 4 `[static]`

Verbose code

3.12.3.14 bool Utils::verbose = false

3.12.3.15 const int Utils::WARNING = 2 `[static]`

Warning code

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

- lib/Utils.h
- lib/Utils.cpp

3.13 Vertex Class Reference

`#include <Vertex.h>`

Classes

- class ConstructionToken

Public Member Functions

- Vertex (ConstructionToken &)

  constructor
- void InsertEdge (Customer &, int)

  Insert an Edge.
- void RemoveEdge (Customer &)

  Remove an edge.
- int GetWeight (Customer &)

  Get the weight of and edge.
- std::map< Customer, Edge > GetEdges () const

  Get the map of the edges

Private Attributes

- std::map< Customer, Edge > edges

3.13.1 Constructor & Destructor Documentation

3.13.1.1 Vertex::Vertex (ConstructionToken &)

constructor

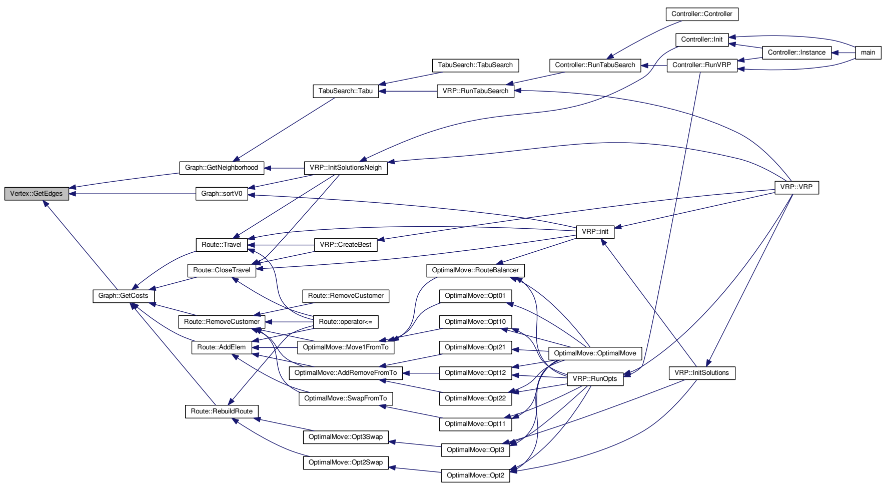
Constructor of Vertex

3.13.2 Member Function Documentation

3.13.2.1 std::map< Customer, Edge > Vertex::GetEdges ( ) const

Get the map of the edges

Here is the caller graph for this function:



3.13.2.2 int Vertex::GetWeight (**Customer** & *c*)

Get the weight of and edge.

Parameters

in	*c*	Customer at the end of the edge

Returns

The weight of the customer

3.13.2.3 void Vertex::InsertEdge (**Customer** & *end_point,* int *weight*)

Insert an Edge.

Insert a weighted edge which end to a customer.

Parameters

in	*end_point*	The destination customer
in	*weight*	The weight of the edge

3.13.2.4 void Vertex::RemoveEdge (Customer & *edge*)

Remove an edge.

Parameters

in	*edge*	The customer which the edge end

3.13.3 Member Data Documentation

3.13.3.1 std::map< Customer, Edge > Vertex::edges [private]

List of all the edges from customer

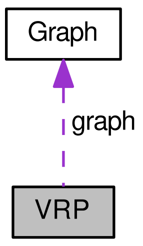
The documentation for this class was generated from the following files:

- lib/Vertex.h
- lib/Vertex.cpp

3.14 VRP Class Reference

```
#include <VRP.h>
```

Collaboration diagram for VRP:



Public Member Functions

- VRP ()
- VRP (const Graph &, const int, const int, const int, const float, const bool, const float, const float)

    constructor
- int InitSolutions ()

    This function creates the routes.
- void CreateBest (std::set< Customer >, std::list< int >, Customer)

    Create a personalize routes.
- int init (int)

    This function creates the routes.
- int InitSolutionsNeigh ()

    This function creates the initial solution.
- void RunTabuSearch (int)
- bool RunOpts (int, bool)

    Run optimal functions.
- int GetTotalCost ()

    Compute the total cost of routes.
- int GetNumberOfCustomers () const

    Number of customers.
- std::list< Route > * GetRoutes ()

    Return the routes.
- std::list< Route > * GetBestRoutes ()

    Return the best configuration of routes.
- bool UpdateBest ()

    Save the best configuration.
- ∼VRP ()

    destructor

Private Member Functions

- bool CheckIntegrity ()

    Check the integrity of the system
- void OrderByCosts ()

    Sort the list of routes by cost.

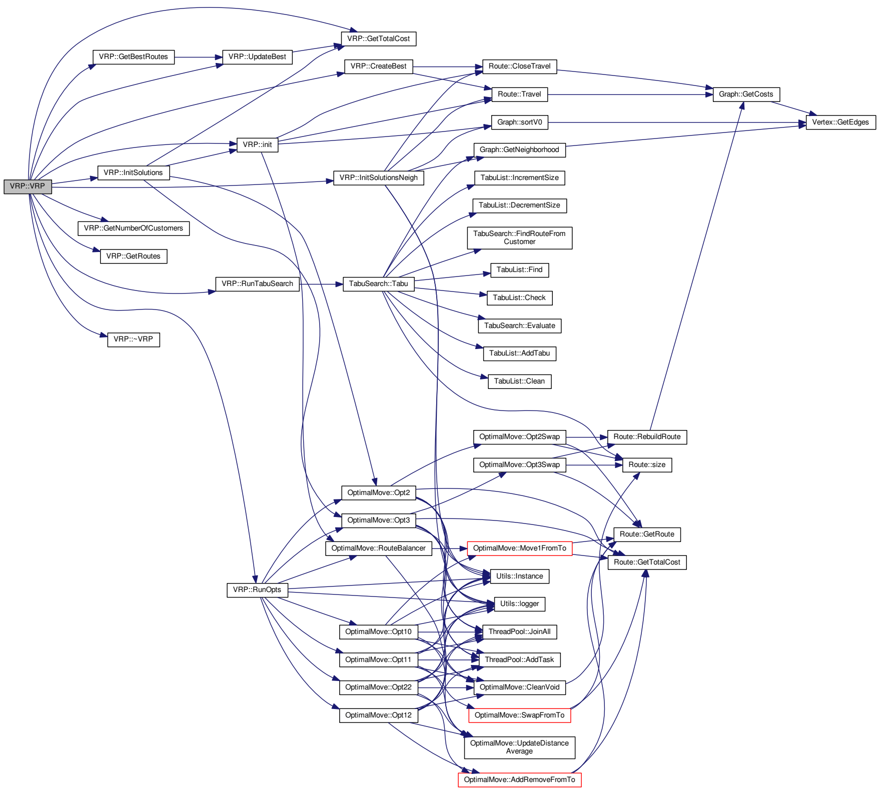
Private Attributes

- Graph graph
- std::list< Route > routes
- int numVertices
- int vehicles
- int capacity
- float workTime
- float costTravel
- float alphaParam
- float averageDistance
- std::list< Route > bestRoutes
- int totalCost

3.14.1 Constructor & Destructor Documentation

3.14.1.1 VRP::VRP ( ) `[inline]`

Here is the call graph for this function:



3.14.1.2 VRP::VRP (const **Graph** & *g*, const int *n*, const int *v*, const int *c*, const float *t*, const bool *flagTime*, const float *costTravel*, const float *alphaParam*)

constructor

Constructor of VRP.

constructor

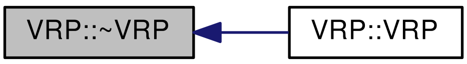
Parameters

in	*g*	The graph generate from the input.
in	*n*	The number of Customers.
in	*v*	The number of vehicles.
in	*c*	The capacity of each vehicle.
in	*t*	The work time of each driver.
in	*flagTime*	If the service time is a constraint.
in	*costTravel*	Cost parameter for each travel.
in	*alphaParam*	Alpha parameter for router evaluation.

3.14.1.3 VRP::∼VRP ( )

destructor

Here is the caller graph for this function:



3.14.2 Member Function Documentation

3.14.2.1 bool VRP::CheckIntegrity ( ) [private]

Check the integrity of the system

Each customer have to be visited only once, this function checks if this constraint is valid in the system. Used for debugging.

Returns

The status of the constraint

3.14.2.2 void VRP::CreateBest (std::set< Customer > *custs*, std::list< int > *best*, Customer *depot*)

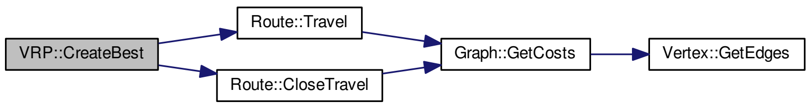
Create a personalize routes.

From a list of indexes of customers creates the route. Created for testing a visualizing the best routes for a known problem.

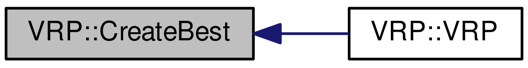
Parameters

`in`	*custs*	Set of all customers
`in`	*best*	List of ordered customer forming the route
`in`	*depot*	The depot

Here is the call graph for this function:



Here is the caller graph for this function:



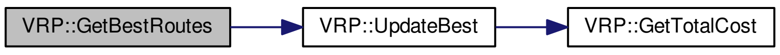
3.14.2.3 std::list< Route > * VRP::GetBestRoutes ( )

Return the best configuration of routes.

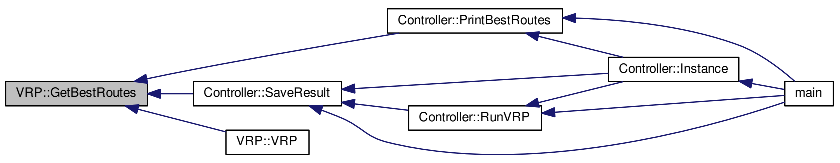
Returns

The pointer to the routes list

Here is the call graph for this function:



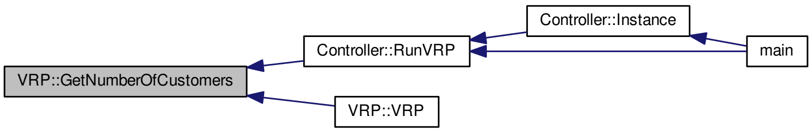
Here is the caller graph for this function:



3.14.2.4 int VRP::GetNumberOfCustomers ( ) const

Number of customers.

Here is the caller graph for this function:



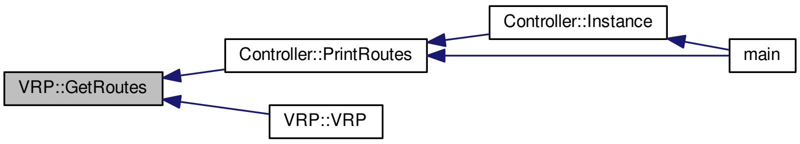
3.14.2.5 std::list< Route > * VRP::GetRoutes ( )

Return the routes.

Returns

The pointer to the routes list

Here is the caller graph for this function:



3.14.2.6 int VRP::GetTotalCost ( )

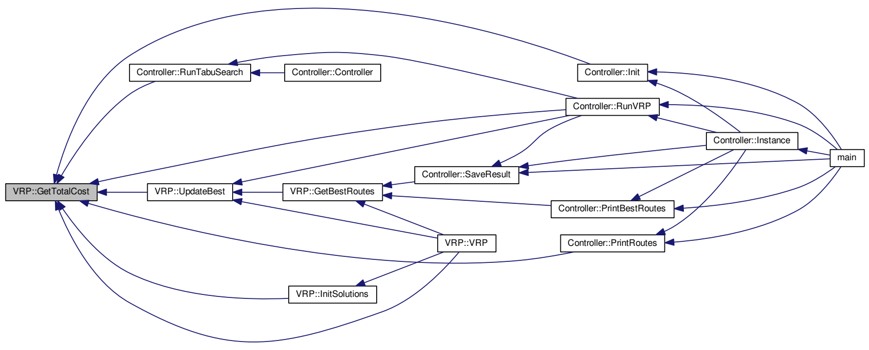
Compute the total cost of routes.

Compute the amount of costs for each route. The cost is used to check improvement on paths.

Returns

The total cost

Here is the caller graph for this function:



3.14.2.7 int VRP::init (int *start*)

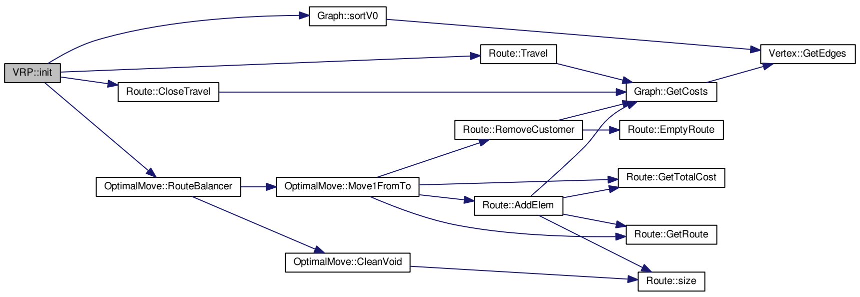
This function creates the routes.

This function creates the initial solution of the algorithm. Customers are sorted in increasing order of distance from the depot. Starting with a random customer the route is created inserting one customer at time. Whenever the insertion of the customer would lead to a violation of the capacity or the working time a new route is created.

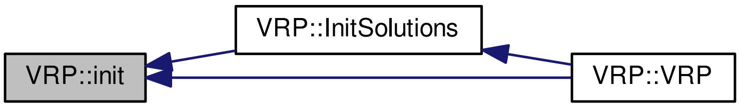
Returns

Error or Warning code.

Here is the call graph for this function:



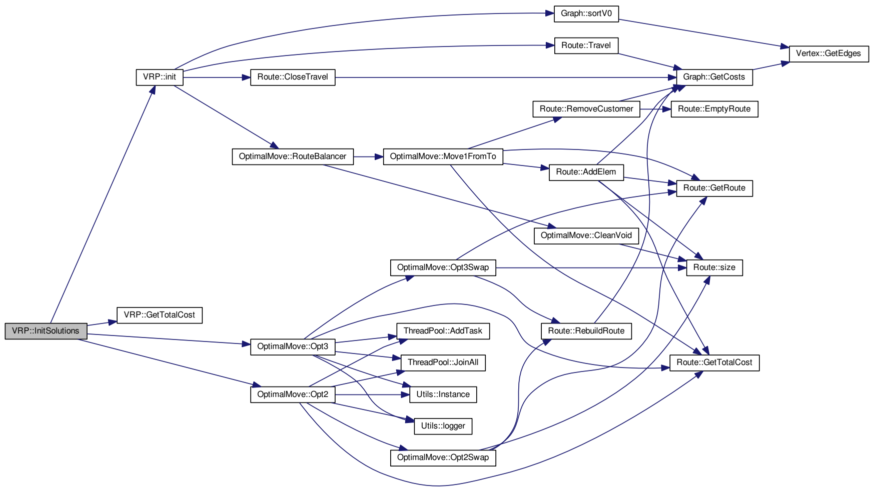
Here is the caller graph for this function:



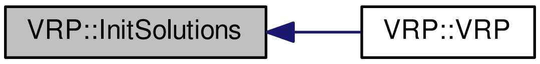
3.14.2.8 int VRP::InitSolutions ( )

This function creates the routes.

Here is the call graph for this function:



Here is the caller graph for this function:



3.14.2.9 int VRP::InitSolutionsNeigh ( )

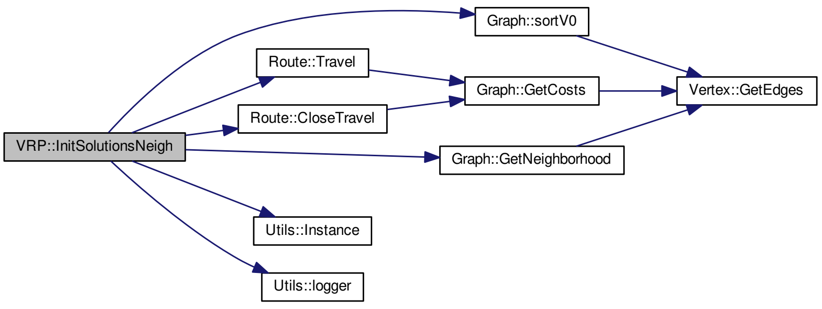
This function creates the initial solution.

This function creates the initial solution of the algorithm. Starting from the depot it creates the routes running an iterated local search for each customer. Whenever the insertion of a customer would lead to a violation of the capacity of the working time a new route is created.

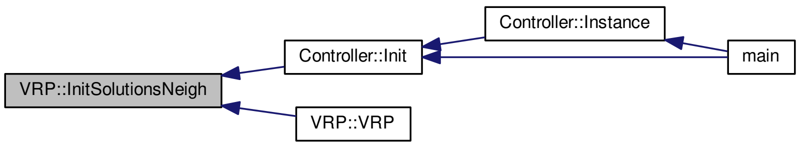
Returns

Error or Warning code.

Here is the call graph for this function:



Here is the caller graph for this function:

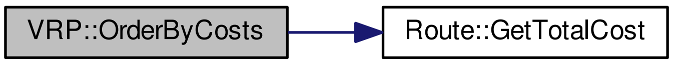


3.14.2.10 void VRP::OrderByCosts ( ) [private]

Sort the list of routes by cost.

This function sort the routes by costs in ascending order.

Here is the call graph for this function:



3.14.2.11 bool VRP::RunOpts (int *times,* bool *flag*)

Run optimal functions.

Runs all the optimal functions to achieve a better optimization of the routes. When the routine do not improve the routes, stops and try to balance the routes.

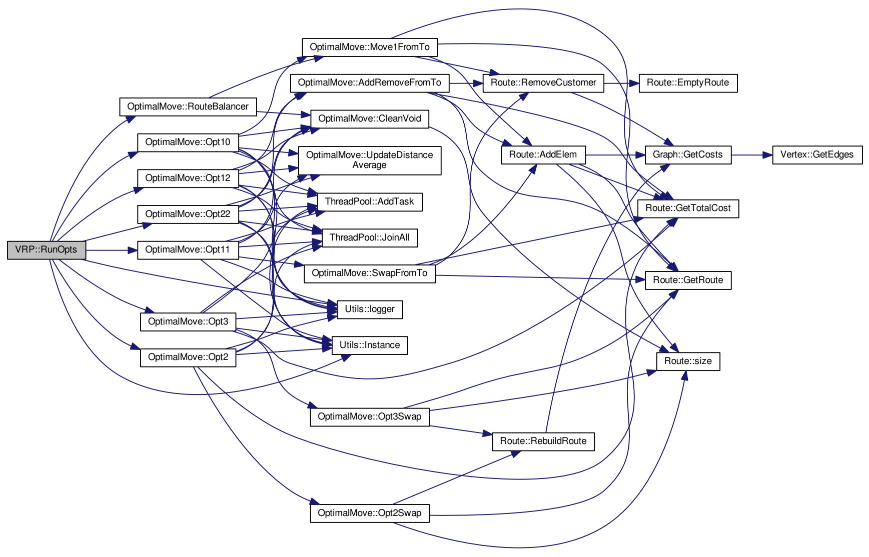
Parameters

`in`	*times*	Number of iteration
`in`	*flag*	Force or not the moves

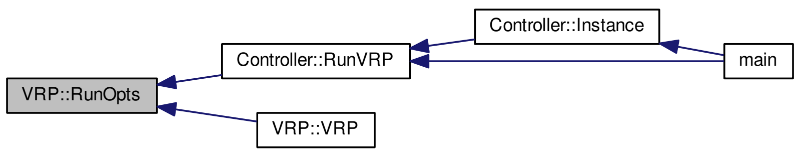
Returns

If the routine made some improvements.

Here is the call graph for this function:

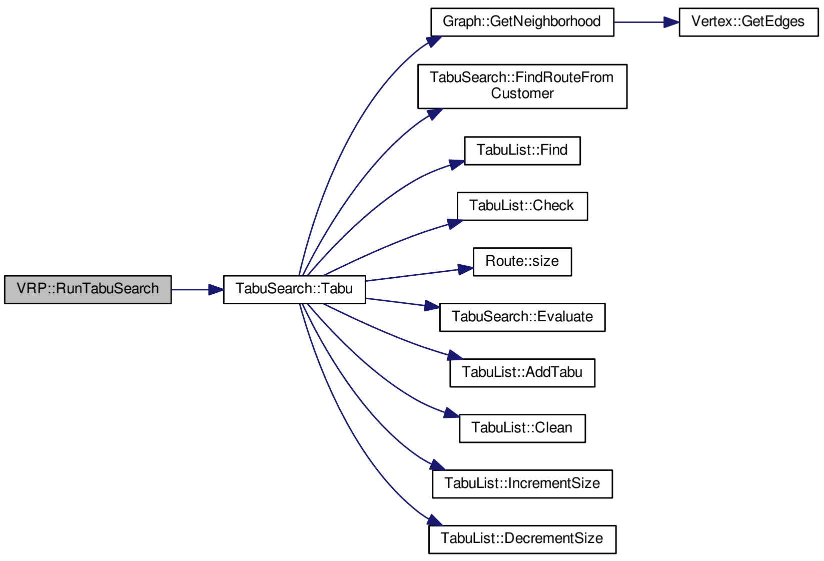


Here is the caller graph for this function:

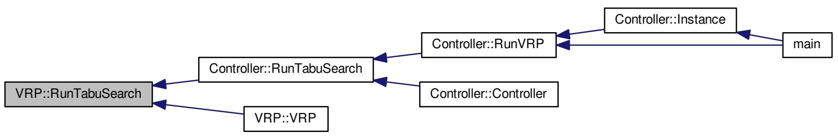


3.14.2.12 void VRP::RunTabuSearch (int *times*)

Here is the call graph for this function:



Here is the caller graph for this function:

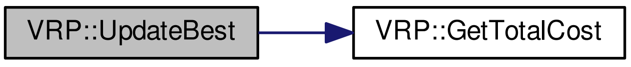


3.14.2.13 bool VRP::UpdateBest ( )

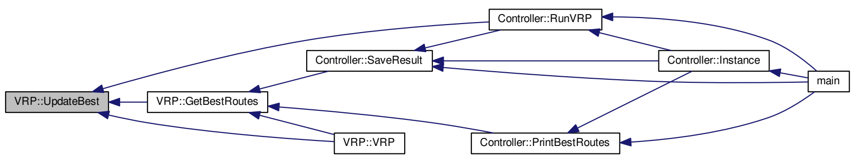
Save the best configuration.

Finds out if the actual configuration is the best and save it.

Here is the call graph for this function:



Here is the caller graph for this function:



3.14.3 Member Data Documentation

3.14.3.1 float VRP::alphaParam [private]

Alpha parameter for route evaluation

3.14.3.2 float VRP::averageDistance [private]

Average distance of all routes

3.14.3.3 std::list<Route> VRP::bestRoutes [private]

The best configuration of routes founded

3.14.3.4 int VRP::capacity [private]

Capacity of each vehicle

3.14.3.5 float VRP::costTravel [private]

Cost parameter for each travel

3.14.3.6 Graph VRP::graph [private]

Graph of customers

3.14.3.7 int VRP::numVertices [private]

Number of customers

3.14.3.8 std::list<Route> VRP::routes [private]

List of all routes

3.14.3.9 int VRP::totalCost [private]

Total cost of routes

3.14.3.10 int VRP::vehicles [private]

Number of vehicles

3.14.3.11 float VRP::workTime [private]

Work time for each driver

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

- lib/VRP.h
- lib/VRP.cpp

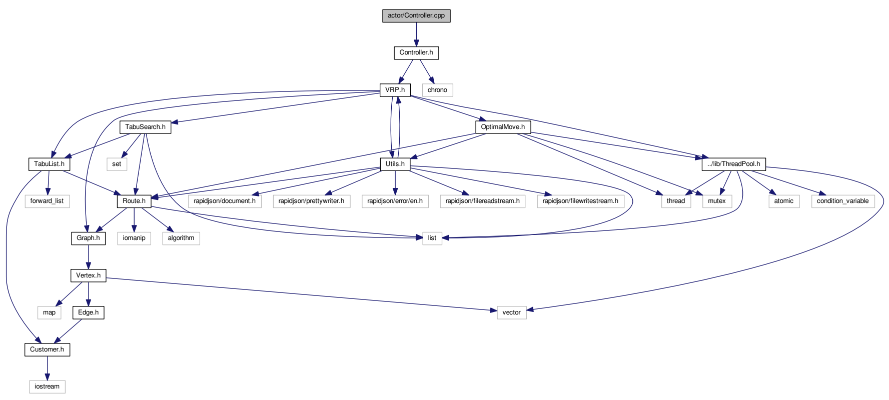
Chapter 4

File Documentation

4.1 actor/Controller.cpp File Reference

```
#include "Controller.h"
```

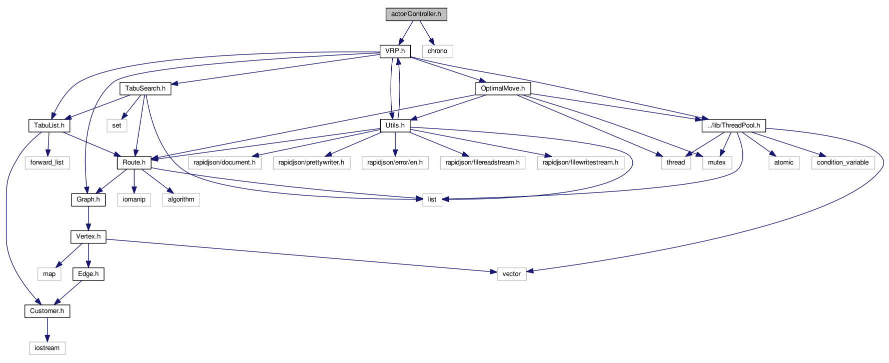
Include dependency graph for Controller.cpp:



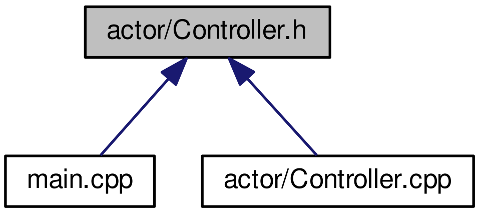
4.2 actor/Controller.h File Reference

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

Include dependency graph for Controller.h:



This graph shows which files directly or indirectly include this file:



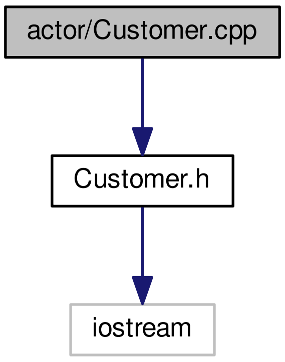
Classes

- class Controller

4.3 actor/Customer.cpp File Reference

```
#include "Customer.h"
```

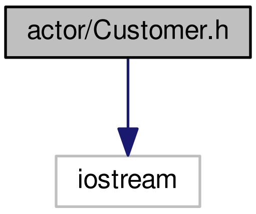
Include dependency graph for Customer.cpp:



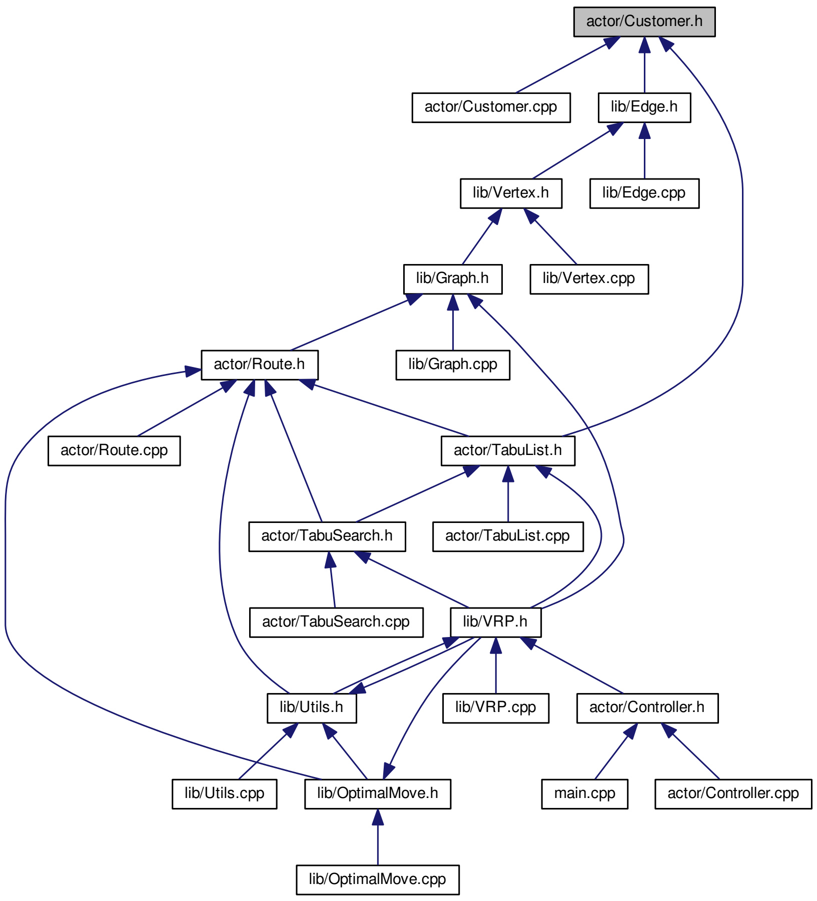
4.4 actor/Customer.h File Reference

```
#include <iostream>
```

Include dependency graph for Customer.h:



This graph shows which files directly or indirectly include this file:



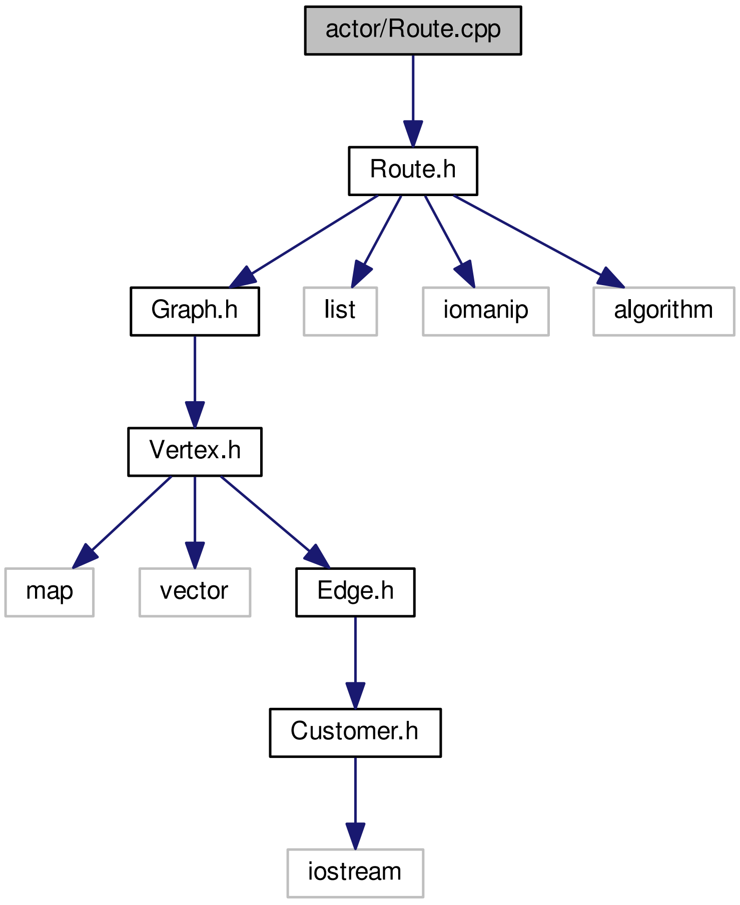
Classes

- class Customer

4.5 actor/Route.cpp File Reference

```
#include "Route.h"
```

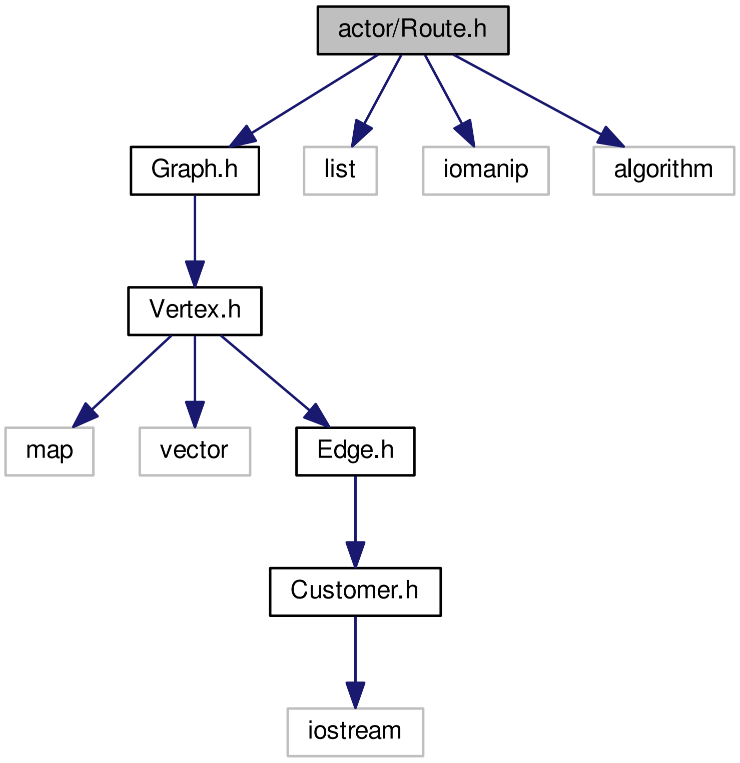
Include dependency graph for Route.cpp:



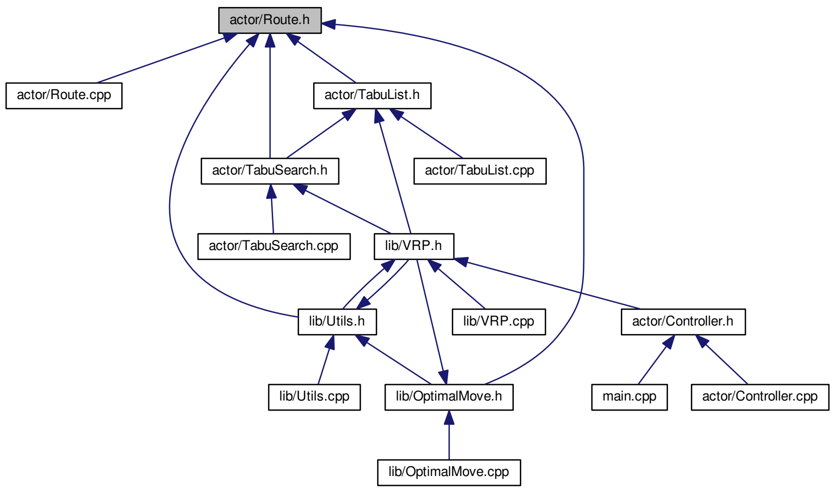
4.6 actor/Route.h File Reference

```
#include "Graph.h"
#include <list>
#include <iomanip>
#include <algorithm>
```

Include dependency graph for Route.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Route

Typedefs

- typedef std::pair< Customer, int > StepType

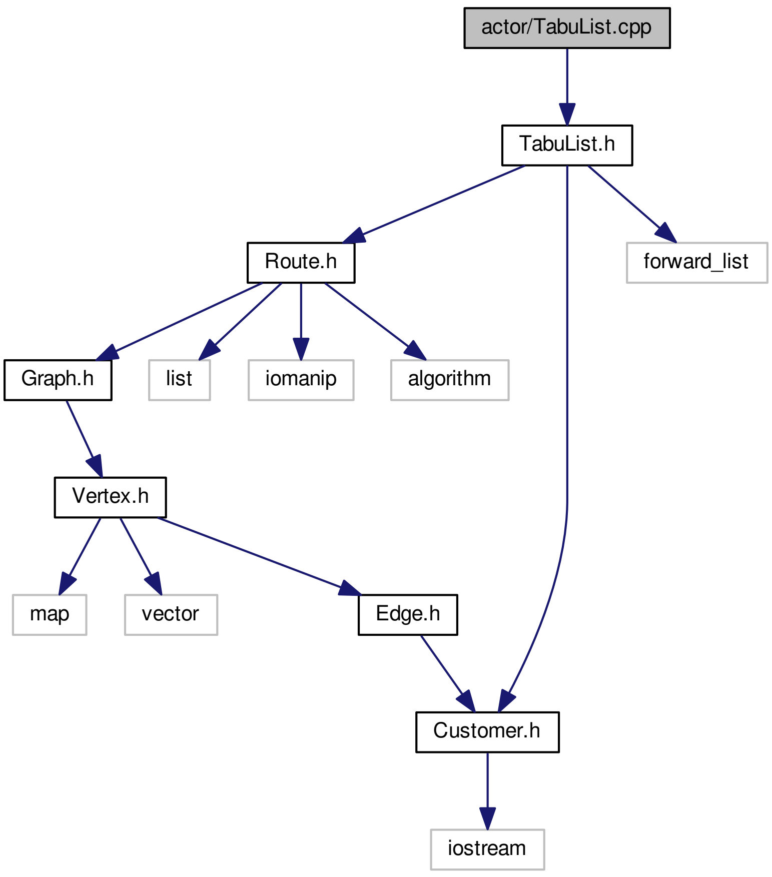
4.6.1 Typedef Documentation

4.6.1.1 typedef std::pair<Customer, int> StepType

4.7 actor/TabuList.cpp File Reference

```
#include "TabuList.h"
```

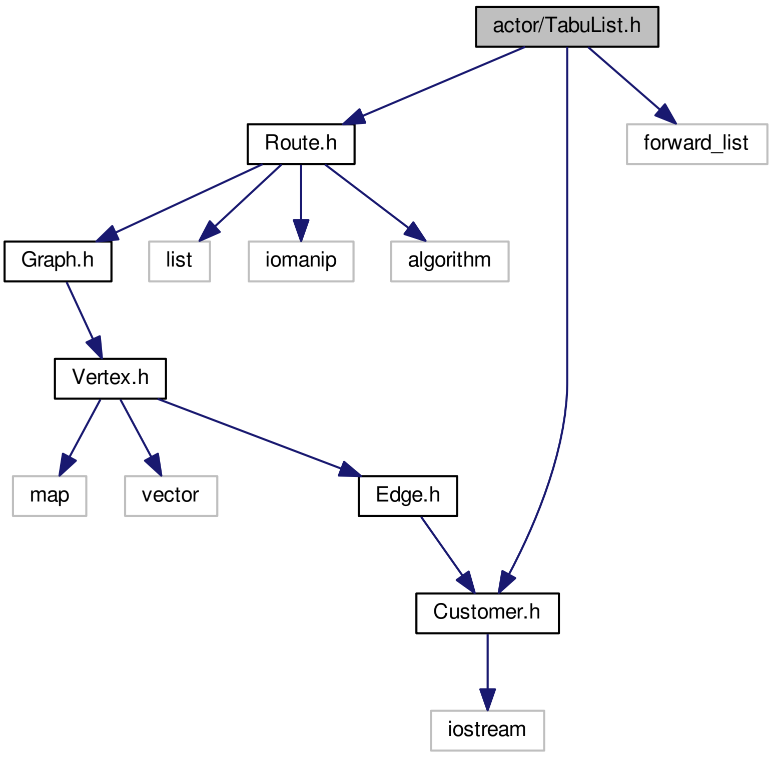
Include dependency graph for TabuList.cpp:



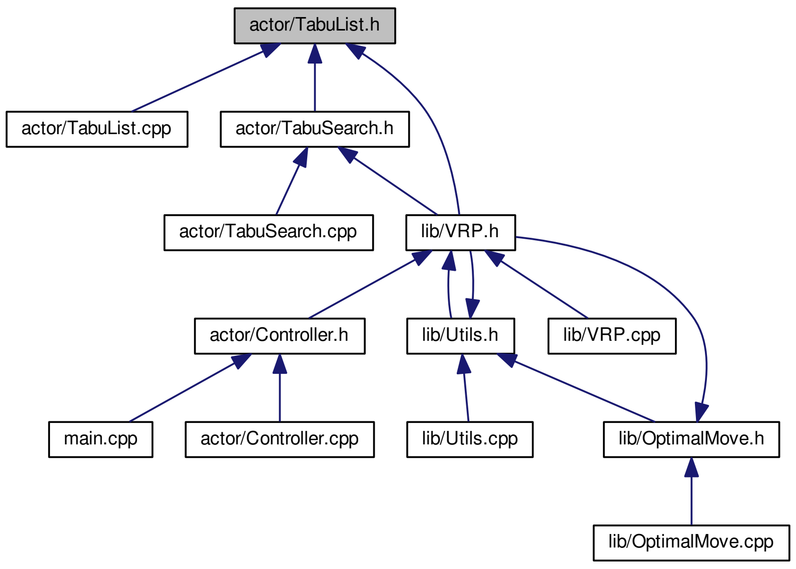
4.8 actor/TabuList.h File Reference

```
#include "Route.h"
#include "Customer.h"
#include <forward_list>
```

Include dependency graph for TabuList.h:



This graph shows which files directly or indirectly include this file:



Classes

- class TabuList

Typedefs

- typedef std::pair< std::pair< Customer, int >, int > Move
- typedef std::pair< Move, float > TabuElement

4.8.1 Typedef Documentation

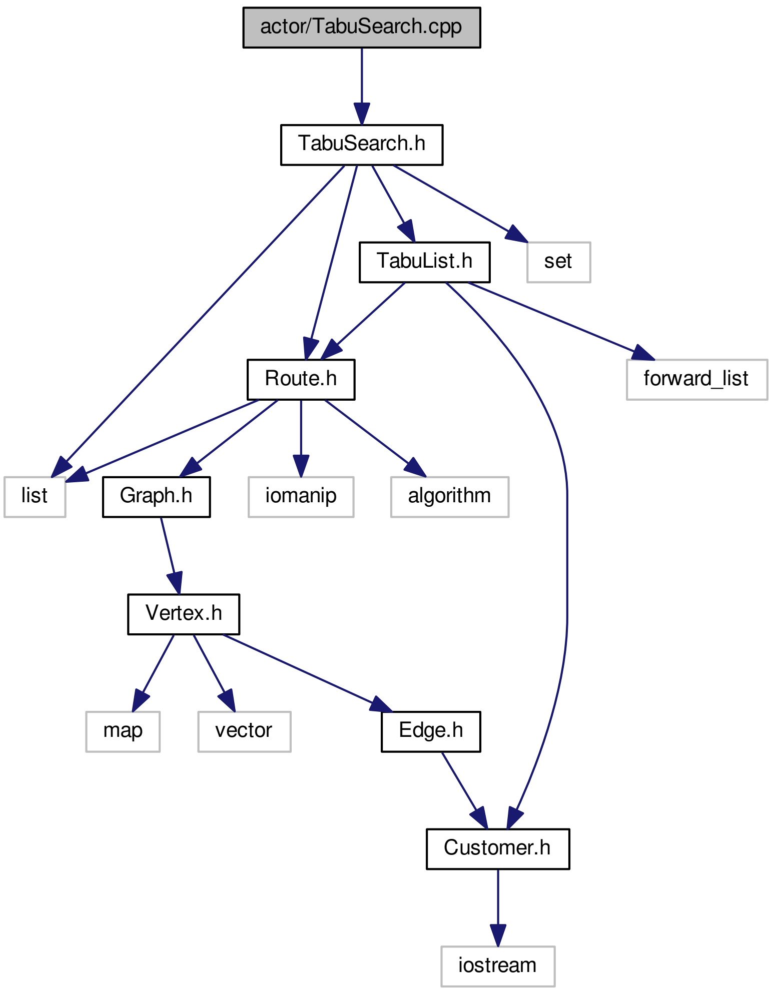
4.8.1.1 typedef std::pair<std::pair<Customer, int>, int> Move

4.8.1.2 typedef std::pair<Move, float> TabuElement

4.9 actor/TabuSearch.cpp File Reference

```
#include "TabuSearch.h"
```

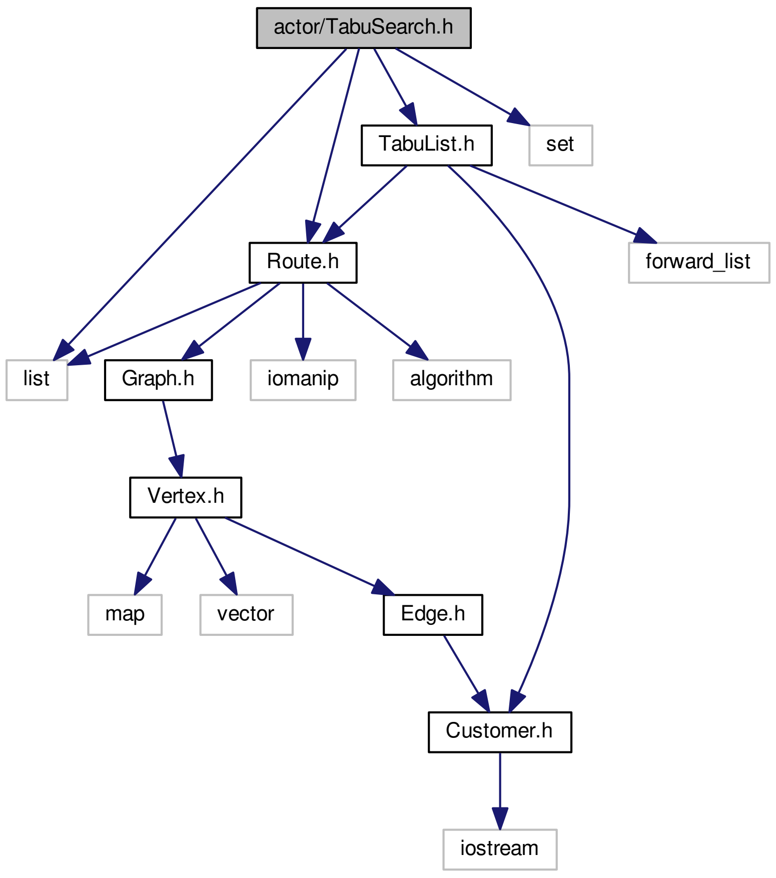
Include dependency graph for TabuSearch.cpp:



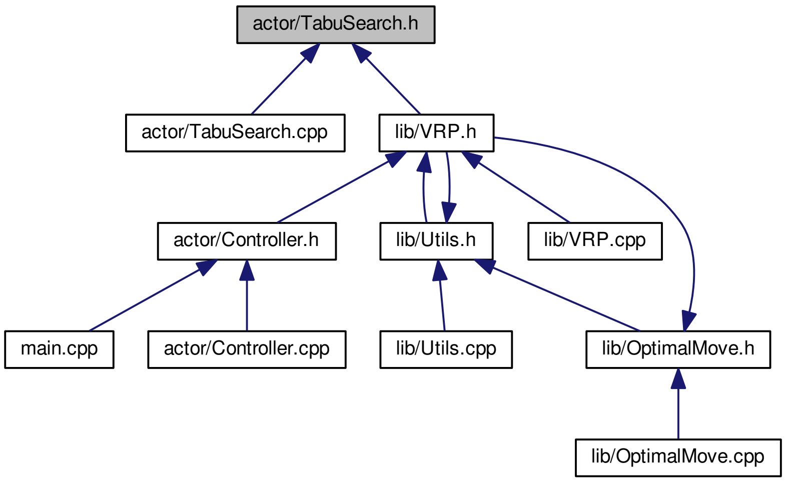
4.10 actor/TabuSearch.h File Reference

```
#include "Route.h"
#include "TabuList.h"
#include <list>
#include <set>
```

Include dependency graph for TabuSearch.h:



This graph shows which files directly or indirectly include this file:



Classes

- class TabuSearch

Typedefs

- typedef std::list< StepType > RouteList
- typedef std::list< Route > Routes

4.10.1 Typedef Documentation

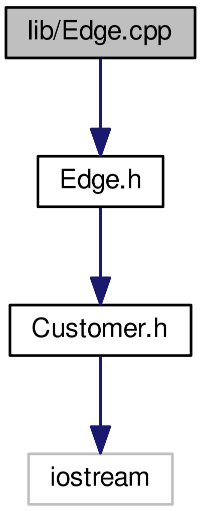
4.10.1.1 typedef std::list<StepType> RouteList

4.10.1.2 typedef std::list<Route> Routes

4.11 lib/Edge.cpp File Reference

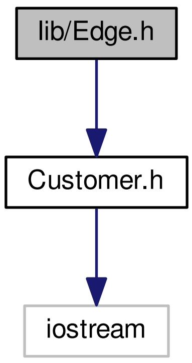
```
#include "Edge.h"
```

Include dependency graph for Edge.cpp:

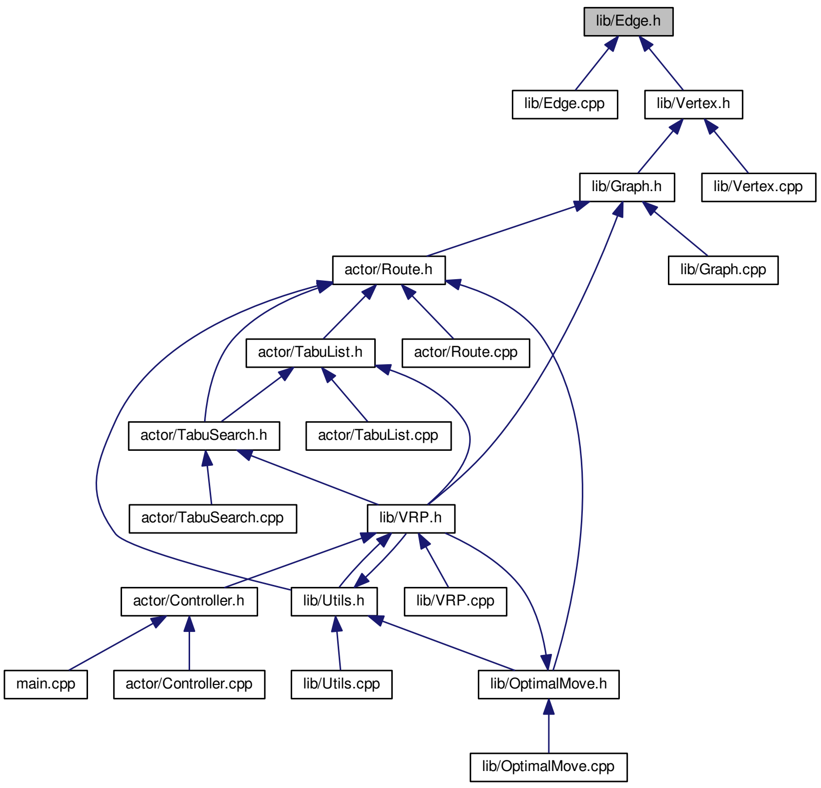


4.12 lib/Edge.h File Reference

```
#include "Customer.h"
```
Include dependency graph for Edge.h:



This graph shows which files directly or indirectly include this file:



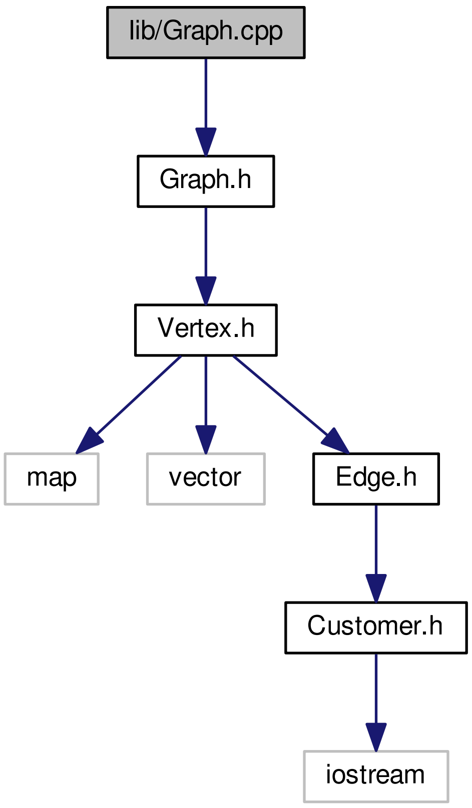
Classes

- class Edge
- class Edge::ConstructionToken

4.13 lib/Graph.cpp File Reference

```
#include "Graph.h"
```

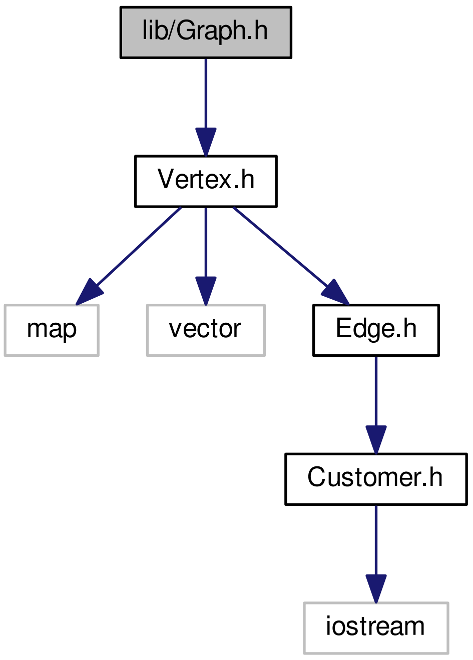
Include dependency graph for Graph.cpp:



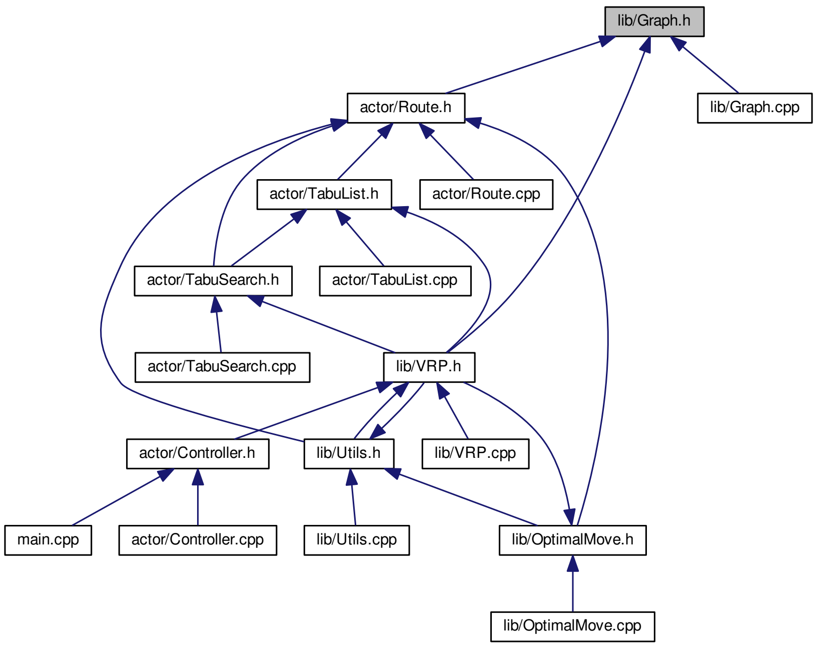
4.14 lib/Graph.h File Reference

```
#include "Vertex.h"
```

Include dependency graph for Graph.h:



This graph shows which files directly or indirectly include this file:



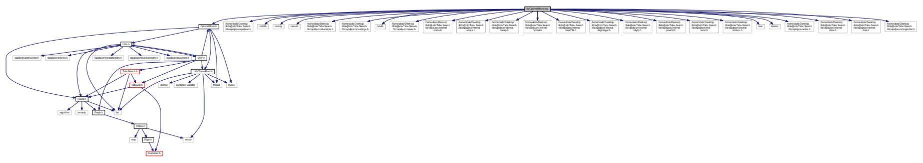
Classes

- class Graph

4.15 lib/OptimalMove.cpp File Reference

```
#include "OptimalMove.h"
```

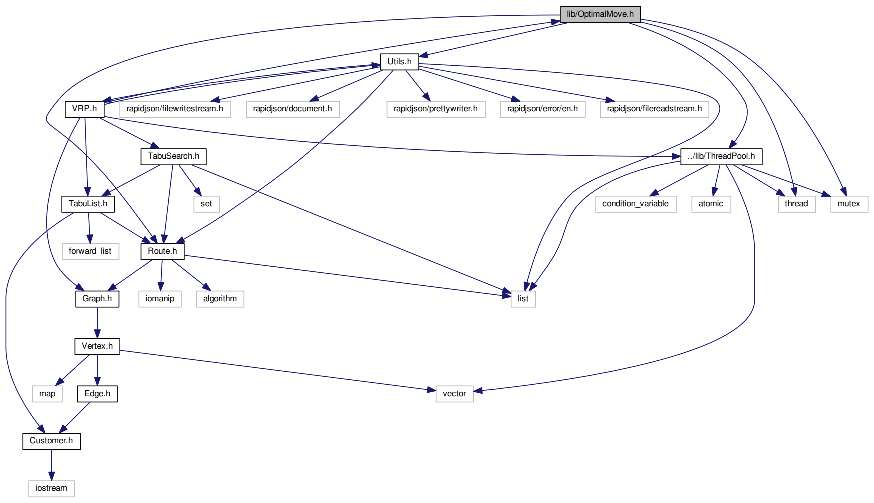
Include dependency graph for OptimalMove.cpp:



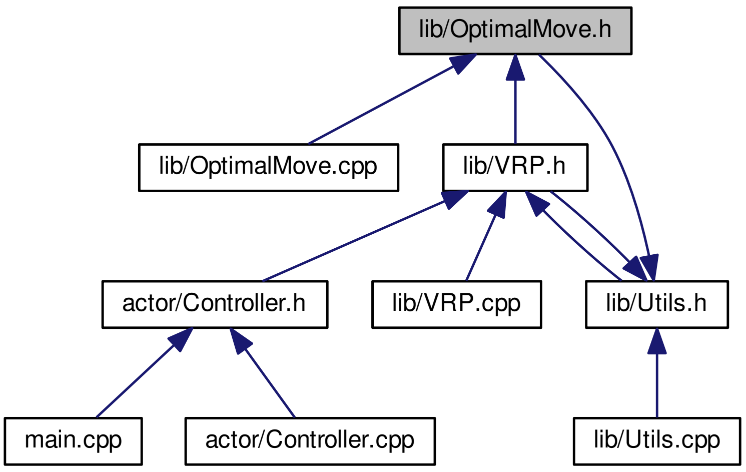
4.16 lib/OptimalMove.h File Reference

```
#include "Route.h"
#include "Utils.h"
#include "../lib/ThreadPool.h"
#include <thread>
#include <mutex>
```

Include dependency graph for OptimalMove.h:



This graph shows which files directly or indirectly include this file:



Classes

- class OptimalMove

Typedefs

- typedef std::list< Route > Routes
- typedef std::pair< std::pair< int, int >, std::pair< Route, Route > > BestResult

Variables

- auto comp

4.16.1 Typedef Documentation

4.16.1.1 typedef std::pair<std::pair<int, int>, std::pair<Route, Route> > BestResult

4.16.1.2 typedef std::list<Route> Routes

4.16.2 Variable Documentation

4.16.2.1 auto comp

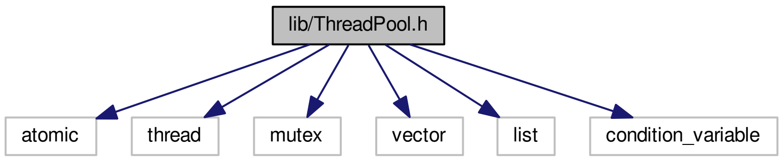
Initial value:

```
= [](const BestResult &l, const BestResult &r)->bool {
    return (l.second.first.GetTotalCost() + l.second.second.GetTotalCost()) <
        (r.second.first.GetTotalCost() + r.second.second.GetTotalCost());
}
```

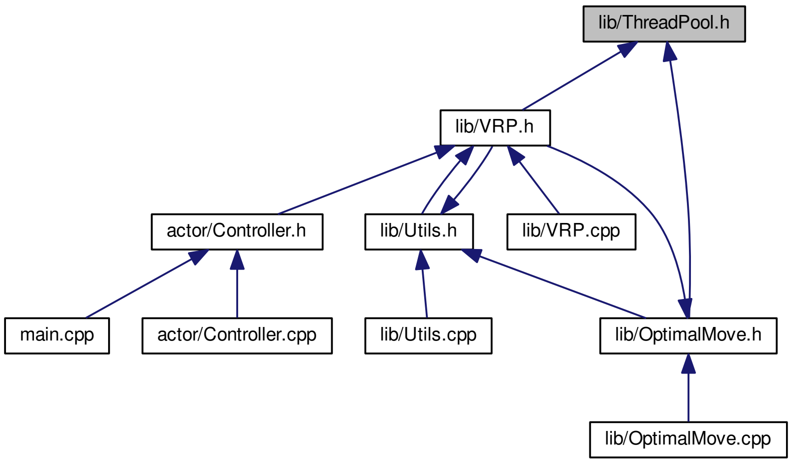
4.17 lib/ThreadPool.h File Reference

```
#include <atomic>
#include <thread>
#include <mutex>
#include <vector>
#include <list>
#include <condition_variable>
```

Include dependency graph for ThreadPool.h:



This graph shows which files directly or indirectly include this file:



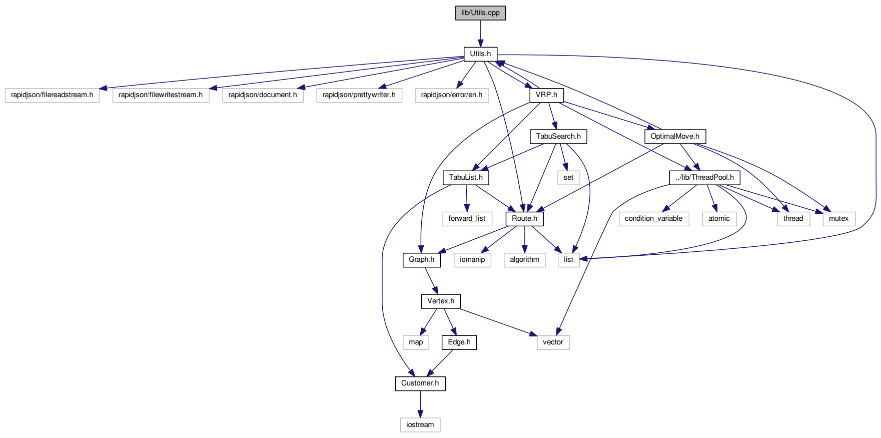
Classes

- class ThreadPool

4.18 lib/Utils.cpp File Reference

```
#include "Utils.h"
```

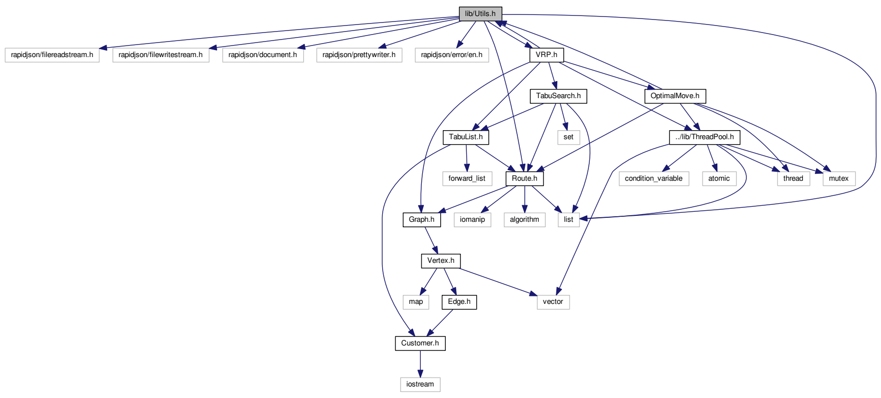
Include dependency graph for Utils.cpp:



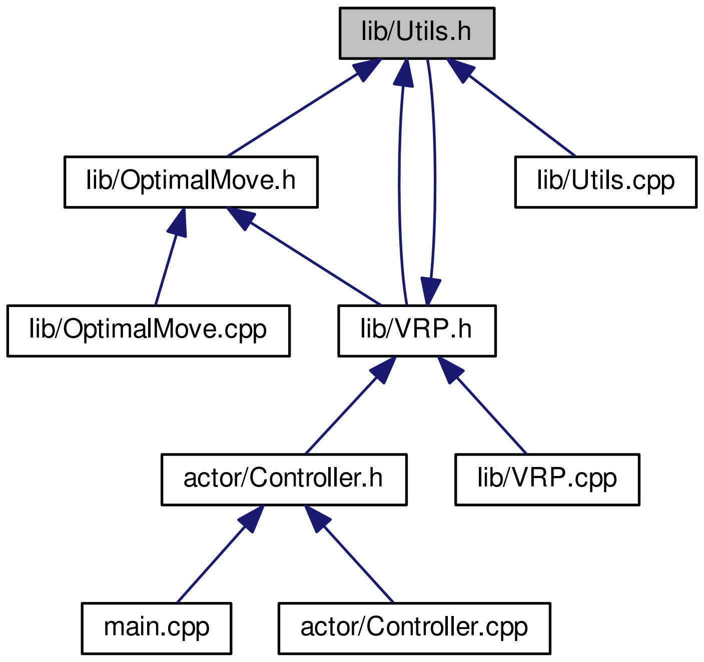
4.19 lib/Utils.h File Reference

```
#include "rapidjson/filereadstream.h"
#include "rapidjson/filewritestream.h"
#include "rapidjson/document.h"
#include "rapidjson/prettywriter.h"
#include "rapidjson/error/en.h"
#include "VRP.h"
#include "Route.h"
#include <list>
```

Include dependency graph for Utils.h:



This graph shows which files directly or indirectly include this file:



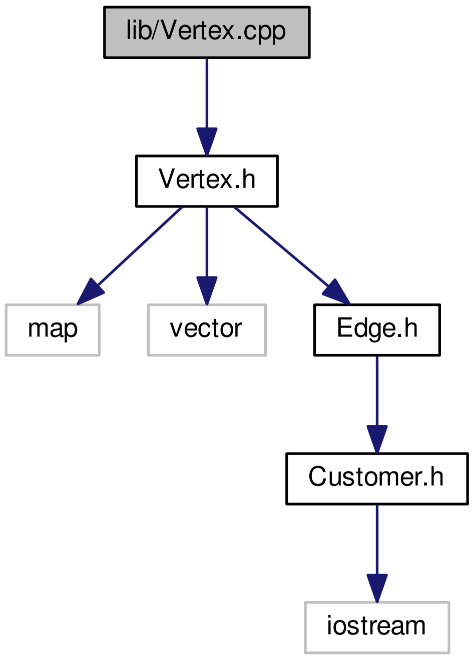
Classes

- class Utils

4.20 lib/Vertex.cpp File Reference

```
#include "Vertex.h"
```

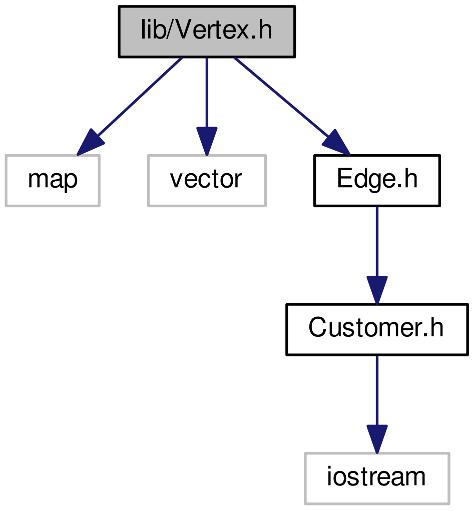
Include dependency graph for Vertex.cpp:



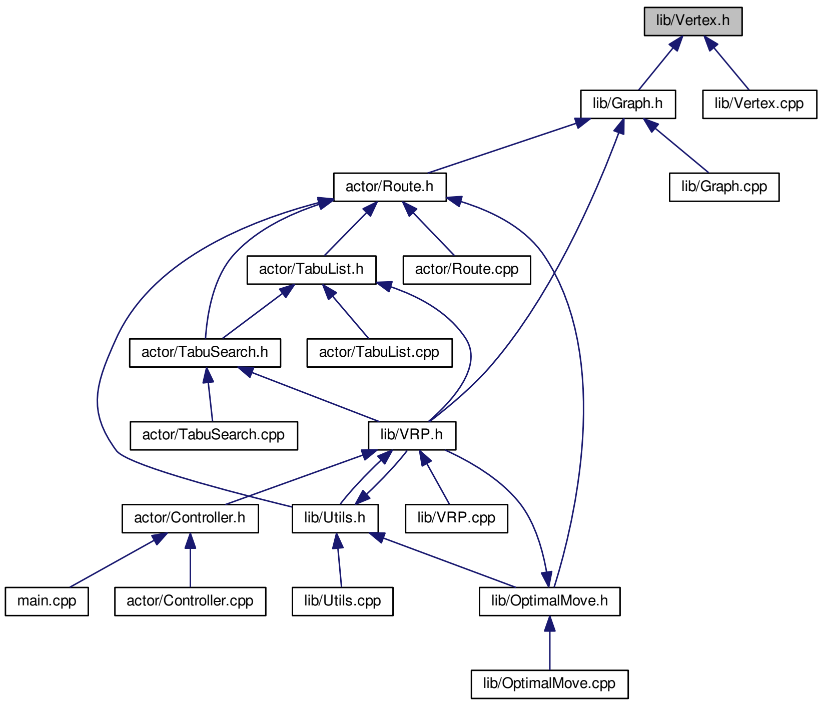
4.21 lib/Vertex.h File Reference

```
#include <map>
#include <vector>
#include "Edge.h"
```

Include dependency graph for Vertex.h:



This graph shows which files directly or indirectly include this file:



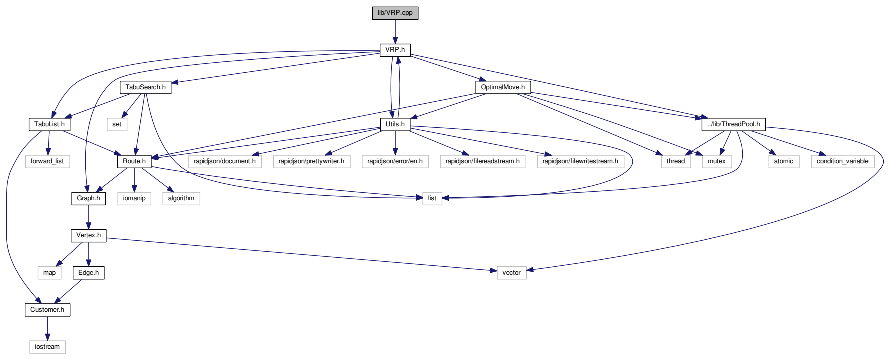
Classes

- class Vertex
- class Vertex::ConstructionToken

4.22 lib/VRP.cpp File Reference

```
#include "VRP.h"
```

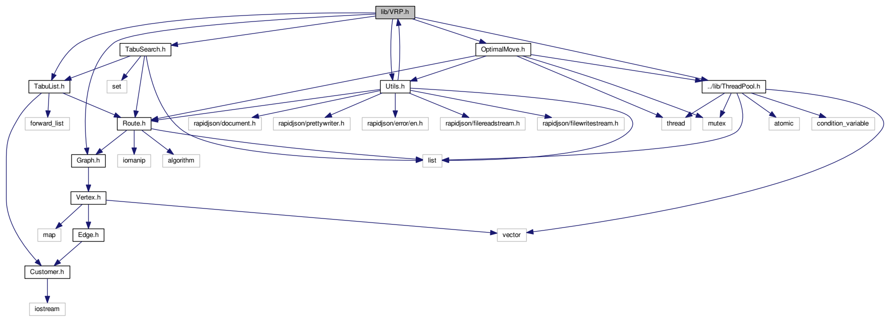
Include dependency graph for VRP.cpp:



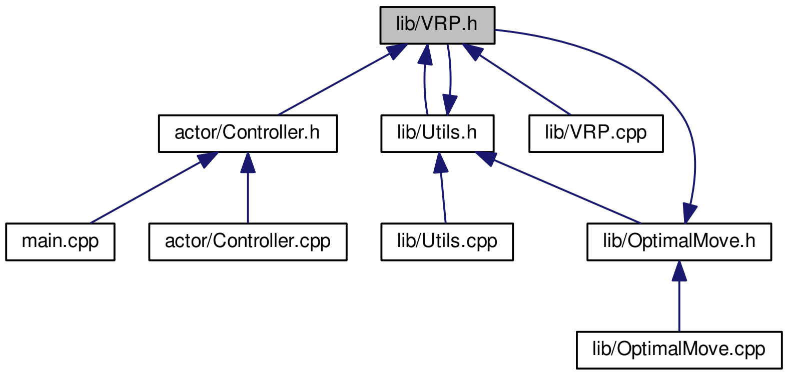
4.23 lib/VRP.h File Reference

```
#include "Graph.h"
#include "Utils.h"
#include "TabuList.h"
#include "OptimalMove.h"
#include "TabuSearch.h"
#include "../lib/ThreadPool.h"
```

Include dependency graph for VRP.h:



This graph shows which files directly or indirectly include this file:



Classes

- class VRP

Typedefs

- typedef std::multimap< int, Customer > Map

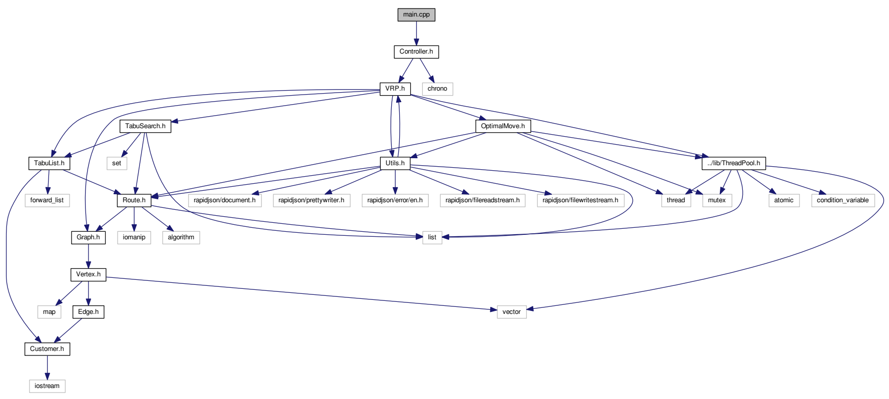
4.23.1 Typedef Documentation

4.23.1.1 typedef std::multimap<int, Customer> Map

4.24 main.cpp File Reference

`#include "Controller.h"`

Include dependency graph for main.cpp:



Macros

- #define TRAVEL_COST 0.3f
- #define ALPHA 0.4f
- #define MAX_TIME_MIN 300

Functions

- int main (int argc, char ∗∗argv)

4.24.1 Macro Definition Documentation

4.24.1.1 #define ALPHA 0.4f

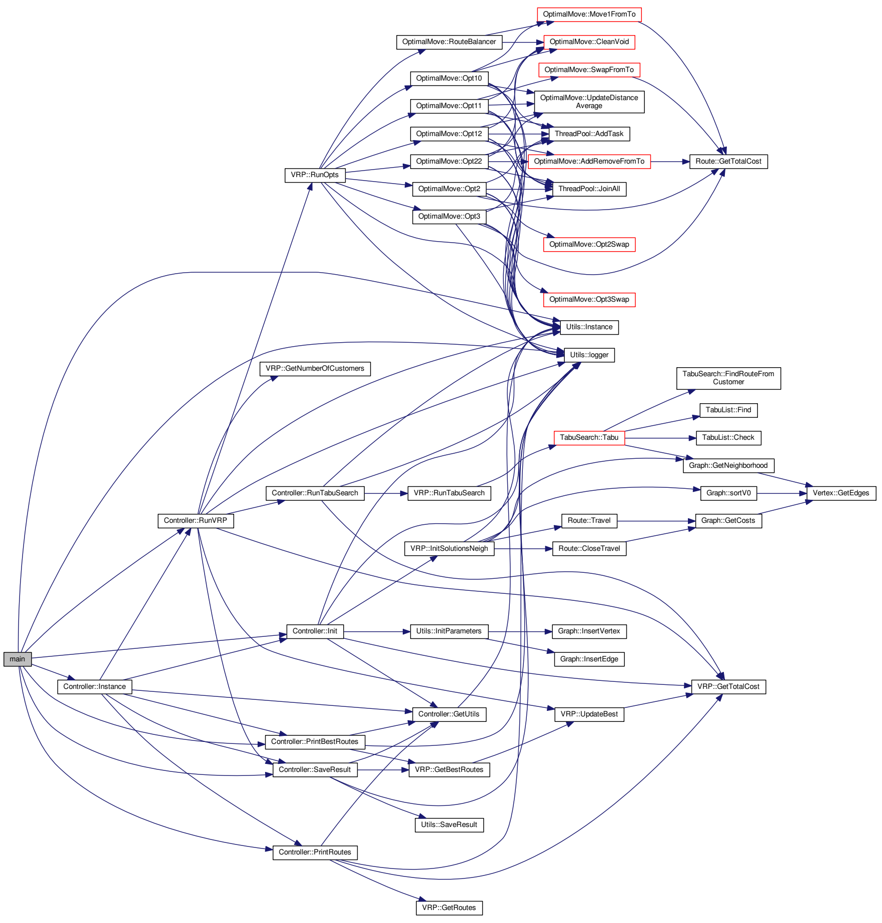
4.24.1.2 #define MAX_TIME_MIN 300

4.24.1.3 #define TRAVEL_COST 0.3f

4.24.2 Function Documentation

4.24.2.1 int main (int *argc,* char ∗∗ *argv*)

Here is the call graph for this function:


OptimalMove::Move1FromTo
OptimalMove::RouteBalancer
OptimalMove::CleanVoid
OptimalMove::SwapFromTo
OptimalMove::Opt10
OptimalMove::UpdateDistance Average
OptimalMove::Opt11
OptimalMove::Opt12
ThreadPool::AddTask
VRP::RunOpts
OptimalMove::Opt22
OptimalMove::AddRemoveFromTo
Route::GetTotalCost
OptimalMove::Opt2
ThreadPool::JoinAll
OptimalMove::Opt3
OptimalMove::Opt2Swap
OptimalMove::Opt3Swap
Utils::Instance
Utils::logger
VRP::GetNumberOfCustomers
TabuSearch::FindRouteFrom Customer
TabuList::Find
TabuSearch::Tabu
TabuList::Check
Graph::GetNeighborhood
Controller::RunTabuSearch
VRP::RunTabuSearch
Graph::sortV0
Vertex::GetEdges
Controller::RunVRP
Route::Travel
Graph::GetCosts
VRP::InitSolutionsNeigh
Route::CloseTravel
Controller::Init
Utils::InitParameters
Graph::InsertVertex
main
Graph::InsertEdge
Controller::Instance
VRP::GetTotalCost
Controller::GetUtils
VRP::UpdateBest
Controller::PrintBestRoutes
Controller::SaveResult
VRP::GetBestRoutes
Utils::SaveResult
Controller::PrintRoutes
VRP::GetRoutes