# Paradiseo: MoLesson1

# Paradiseo-MO Lesson 1: Implementing a Hill Climbing

In this lesson, all the source code is in */tutorial/Lesson1* and the executable in *build/tutorial/Lesson1*.

## Introduction

Before beginning this lesson, be sure that the paradiseo-mo package has been compiled (see the README file in the paradiseo-mo directory if you do not know how compile the paradiseo-mo package).

Furthermore, it can be interesting to have documentation opened (open the file */doc/html/index.html*).

Open a terminal and from the paradiseo-mo directory go where the executable of the lesson 1 is (*/build/tutorial/Lesson1*).

A hill climbing has been already designed and compiled. It works on the Traveling Salesman Problem (TSP). You can test it with an instance that is in the "benchs" directory (*build/tutorial/examples/tsp/benchs*). For example, you can type :

```
./hill_climbing ../examples/tsp/benchs/eil101.tsp
```

Here, an example of what you can see on your screen after executing the command :

```
>> Loading [benchs/eil101.tsp]

[From] -3435 101 36 55 54 65 50 80 14 27 46 15 93 58 78 47 99 64 68 49 72 81 82 76 41 16 66 30 29 83 95
71 38 35 19 5 63 26 43 96 79 52 59 69 7 56 37 3 32 28 9 13 1 91 31 73 11 40 62 17 24 45 88 6 85 22 18 20 44
94 12 33 60 89 74 42 97 21 100 57 77 92 98 51 2 25 75 10 86 8 23 67 70 34 90 87 48 53 0 39 4 84 61

[To] -708 101 36 97 91 58 95 98 92 84 99 90 43 13 37 85 15 60 16 83 4 59 82 44 7 45 35 48 63 10 18 46 47
81 17 51 26 27 25 39 57 52 100 88 5 93 94 12 1 56 86 96 41 42 14 40 21 73 71 72 20 3 55 74 22 66 38 24 54
53 23 28 78 2 76 67 79 11 75 68 0 49 32 80 77 33 50 8 34 70 64 65 19 29 69 31 89 62 9 30 87 6 61
```

You have launched a hill climbing on the eil101 TSP instance.

## A Hill Climbing for the TSP

### Format of 'hill_climbing.cpp':

Now open the 'hill_climbing.cpp' file (in *tutorial/Lesson1*). You can see 3 main parts in the file:

- ***comments*** (from line 1 to line 10): it contains information about the file. **NB: if you have have any problems, use the address given in these commentaries.**
- ***include***

(from line 12 to line 22): it indicates which files are needed to be included to the program. The file names beginning by "mo" correspond to files which are the 'src' directory of the paradiseo-mo package and the others to files specifically designed for making the hill climbing for the TSP (these files are in the directory 'tutorial/examples/tsp/src').

- ***main*** (from line 24 to the end): it contains the hill climbing main code.

**NB : during this lesson, if you want to use an object that is not indicated in the include part, you need to include the corresponding file before using it.**

**moHC class:**

In order to better understand this lesson, open the documentation generated by doxygen (file *doc/html/index.html*). On the top of the opened window, click on "Classes", then on "Class Hierarchy" and to finish on "moHC<M>". You have now the documentation of the moHC class which describes a hill climbing in the paradiseo-mo package.

In the moHC class, there are two constructors, we are going to use the first one, click on it in the documentation (click on the first "moHC" under "Public Member Functions").

You have now the description of the constructor and its 5 parameters. The first four parameters are paradiseo-mo objects (beginning by 'mo') and the fifth is a paradiseo-eo object (beginning by 'eo').

You have certainly noticed a <M> that appears everywhere... it correspond to the notion of move (moMove). A moHC is a generic class (a template) that can be used on what you want according to the fact you give a description for <M>.

In this example, <M> corresponds to 'TwoOpt'. It is a well know operator for the TSP. But how has been designed the TwoOpt type ??? It depends on how a TSP solution is represented.

## Route and TwoOpt:

In our example, a TSP solution is a Route, more precisely it is a vector of unsigned and it can have a fitness represented by a float (described in 'route.h').

TwoOpt
describe a move that can be applied to a TSP solution, in our case a Route... that is why in the 'two_opt.h' file, you can see '... public moMove<Route> ...'. So 'two_opt.h' describes a TwoOpt move and 'two_opt.cpp' explains how applies a TwoOpt move to a route. In our case, a TwoOpt object has only two parameters that corresponds to the index of 2 cities. Apply of TwoOpt move on a TSP solution (a Route in our example) corresponding to swap all the cities between these two indexes.

Only when the solution representation is chosen, the move can be designed. And only when the move is designed, we can design the objects needed by the moHC constructor.

## Constructor parameters:

As you have already noticed, 4 parameters of the constructor are based on a <M> type, so they are the classes that have to be designed according to the move representation.

- ***moMoveInit***
: it only corresponds to the manner to give initial values to the move parameters. In our case, the object TwoOptInit is a moMoveInit for the TwoOpt move ('...public moMoveInit<TwoOpt>' in 'two_opt_init.h'). Its application is very simple: the first index is 0 and the second is 2 (see 'two_opt_init.cpp').

- ***moNextMove***
  : this type of object has to indicate if another move can be generated and if the answer is yes, it has to generate it. In our example, the object TwoOptNext is a moNextMove for the TwoOpt move ('...public moNextMove<TwoOpt>' in two_opt_next.h'). Its application correspond to increase the value of the second index by one and if the value correspond to the bound of the vector, the first index is increased and the second index = first index + 2.

- ***eoEvalFunc*** : as indicated earlier, it is not an object of paradiseo-mo but it is an object of paradiseo-eo. But the using is the same. An eoEvalFunc allows to evaluate a <EOT>; in our case, a <EOT> is a Route. So an object RouteEval
  is a eoEvalFunc for a Route ('...public eoEvalFunc<Route>' in route_eval.h). When a Route is given to a RouveEval
  object, the Route fitness is computed. Its application in our example corresponds to the sum of the distance between each consecutive city of the Route, it is a negative value (see 'route_eval.cpp').

- ***moIncrEval***
  : this object corresponds to the evaluation of a solution without knowing it (???). More precisely, we have a evaluated solution and a move; and we want to know the fitness of the solution gained when the move is applied. The interest is to find the fitness without completely evaluating the new Route (as it is made in an eoEvalFunc). In our case, a TwopOptIncrEval is a moIncrEval for the TwoOpt move ('...public moIncrMove<TwoOpt>' in 'two_opt_incr_eval.h'). Its application allows to know the fitness of a new solution without apply the move.

- ***moMoveSelect***
  : this object describes how the moves are selected,i.e. how a move is considered to be better than another one. The paradiseo-mo packages proposes three examples of moMoveSelect that can be directly used moBestImprSelect, moFirstImprSelect and moRandImprSelect; see the documentation) or you can also designed your own selector ('...public moMoveSelect<MyMove>'). For our example, we have used the moBestImprSelect object.

## Instantiate our hill climbing:

Now, we have all the parameters ready, it is also important to have an initial solution (a Route in our case) to launch the hill climbing on it (lines 37 to 40):

*Route route ;*

*RouteInit init ;*

*init (route) ;*

The RouteInit object is a eoInit object used with the Route type ('...public eoInit<Route>' in route_init.h). It is an interesting object in order to initialise solution (see paradiseo-eo documentation for eoInit).

The initial solution has to be evaluated, so we used our evaluation function (line 42 and 43):

*RouteEval full_eval ; full_eval (route) ;*

The remaining objects are prepared (lines 45 to 58):

*TwoOptInit two_opt_init ;*

*TwoOptNext two_opt_next ;*

*TwoOptIncrEval two_opt_incr_eval ;*

*moBestImprSelect <TwoOpt> two_opt_select ;*

The moHC is finally instantiated (line 60):

*moHC <TwoOpt> hill_climbing (two_opt_init, two_opt_next, two_opt_incr_eval, two_opt_select, full_eval) ;*

It can be launched on our initial solution (line 61):

*hill_climbing (route) ;*

### Modify the current hill climbing:

The easiest way to modify the behaviour of this program is to replace generic classes by another generic classes. In our case, this concerns the moBestImprSelect object. You can replace it by a moFirstImprSelect object.

Save your modification and now type 'make' at the prompt of the Lesson1 directory (*build/tutorial/Lesson1*).

If you do not have any errors, you can test the new program as you have made earlier:

*./hill_climbing ../examples/tsp/benchs/eil101.tsp*

It works ??? yes, **you are unbelievable**.

We are going to arrive at the end of this lesson... What have we learned ?

## Epilogue

In order to design a hill climbing with the paradiseo-mo packages, you need to chose how a solution of our problem is represented. Then you need to create a object which is a move (moMove) that works with our solution representation ('...public eoMove<mySolutionRepresentation>').

When the move representation is designed, you need to instantiate a moHC object. First, choose which constructor you want to use and watch carefully the description of each parameters. For each of them, thanks to the documentation, search if it already exists a generic implementation that has a behaviour you would like to use in order to not design something that already exists (example of the 'moSelect' objects). If no interesting generic implementation can be used in your case, design your own according to the type of object you want to use (example of the moInit, moNext and moIncrEval objects).

Finally, create an initial solution, evaluate it, prepare all the moHC parameters, instantiate the moHC and launch it on your initial solution.

Good Luck.

## What is next ???

You want to try another solution based heuristic, try the lesson 2 to learn to design a ***tabu search*** algorithm or the lesson 3 to learn to design a ***simulated annealing***.