From Paradiseo

Paradiseo: MoLesson4

Paradiseo-MO Lesson 4: Implementing a Iterated Local Search

To be able to follow this lesson without any problems, do the lesson 1 first.

In this lesson, all the source code is in */tutorial/Lesson4* and the executable in *build/tutorial/Lesson4*.

According to the color and the style used, an object is:

- ***object*** <=> a generic template (abstract).
- **object** <=> an implementation of a generic template class (ready-to-use class).
- ***object*** <=> a template that the user need to implement for his problem.
- **object** <=> an implementation (of a blue or red template) that the user have designed for his problem.

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 opened the documentation(file */doc/html/index.html*).

NB: for the Unix user, type also the command *make install* to add the *benchs* directory in the right localization.

Open a terminal and from the ParadisEO-MO directory go where the executable of the lesson 1 is (*/build/tutorial/Lesson4*).

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 :

./iterated_local_search ../examples/tsp/benchs/berlin52.tsp

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

>> Loading [benchs/berlin52.tsp]

[From] -29414 52 1 20 40 48 9 27 13 22 5 28 24 29 21 26 44 38 33 37 45 31 42 18 12 3 14 36 30 6 51 32 17 11 0 34 4 10 43 50 16 2 23 35 19 46 49 39 25 15 41 8 7 47

[To] -7715 52 0 21 17 30 22 19 49 15 28 29 1 6 41 20 16 2 44 18 40 7 8 9 42 32 50 10 51 13 12 46 25 26 27 11 24 3 5 14 4 23 47 37 39 38 36 45 43 33 34 35 48 31

You have launched an iterated local search on the berlin52 TSP instance.

A Iterated Local Search for the TSP

Format of 'iterated_local_search.cpp':

Now open the 'iterated_local_search.cpp' file (in *tutorial/Lesson4*). 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 24): 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 iterated local search for the TSP (these files are in the directory 'tutorial/examples/tsp/src').
- ***main*** (from line 24 to the end): it contains the iterated local search 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.

moILS 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 "moILS<M>". You have now the documentation of the **moILS** class which describes a iterated local search in the ParadisEO-MO package.

One of the particularities of a **moILS** is to use a ***moAlgo***. Thus, you are going to use a solution based metaheuristic in a solution based metaheuristic (incredible...).

The first constructor of a **moILS** is the most generic one. It takes five parameters including a ***moAlgo***. It can be useful if you have designed your own ***moAlgo*** and you would like to use it in a **moILS**. If you want to use one of the three other ***moAlgo*** available, simply use the second (using **moHC**), the third (using a **moTS**)or the fourth constructor of **moILS** respectively (using **moSA**).

In our example, we are going to use the second constructor and so use a **moHC** in our **moILS**. In the documentation, click on the second "moILS" under "Public Member Functions").

You have now the description of the constructor and its 8 parameters. Four parameters are dedicated to the internal **moHC** (__move_init, __next_move, __incr_eval and __move_select), one is used by the **moHC** and the **moILS** (__full_eval) and the last 3 are dedicated to the **moILS**. We are going to describe only the 3 parameters dedicated to the **moILS**, if you want any information about the other parameters, go to the **lesson 1**.

- ***moSolContinue***: it is the stopping criterion for the **moILS** algorithm. There are 4 available implementation of this object:
  - **moFitSolContinue**: a fitness threshold is gained.
  - **moGenSolContinue**: a given number of iterations is reached.
  - **moNoFitImprSolContinue**: a given number of iteration are made without fitness improvement.
  - **moSteadyFitSolContinue**: a given number of iterations is reached then the stopping criterion is validated when a given number of iterations are made without improvement. It is a combination of a **moGenSolContinue** and a **moNoFitImprSolContinue**. It is inspired of the eo object **eoSteadyFitContinue**.

- ***moComparator***: it allows to describe how a solution is better than an other. In our case, how a **Route** is better than an other. We use a trivial implementation of this template, it is the **moFitComparator**: a **Route** is better than an other if its fitness is better.

- ***eoMonOp*** : this object has to perturbate a solution. In our case, it has to make some modification on a **Route**, we use a well-known TSP operator: the **CitySwap**. Two indexes of the current **Route** (two cities) are simply swapped.

Instantiate our iterated local search:

Now, we have all the parameters ready, it is also important to have an initial solution (a **Route** in our case) to launch the iterated local search 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 59):

TwoOptInit two_opt_init ;

TwoOptNext two_opt_next ;

TwoOptIncrEval two_opt_incr_eval ;

moBestImprSelect <TwoOpt> two_opt_select ;

moBestImprSelect <TwoOpt> two_opt_select ;

moGenSolContinue <Route> cont (1000) ;

moFitComparator<Route> comparator;

CitySwap perturbation;

The **moILS** is finally instantiated (line 61):

moILS<TwoOpt> iterated_local_search (two_opt_init, two_opt_next, two_opt_incr_eval, two_opt_select, cont, comparator, perturbation, full_eval) ;

It can be launched on our initial solution (line 63):

iterated_local_search (route) ;

Modify the current iterated local search:

The best modification to made, if you have time, is to use the third (using **moTS**) or the fourth constructor of **moILS** (using **moSA**).

Make yours modifications, then save them and type 'make' at the prompt of the Lesson4 directory (*build/tutorial/Lesson4*).

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

./hill_climbing ../examples/tsp/benchs/berlin52.tsp

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

Compare your results with the first version of your iterated local search.

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

Epilogue

In order to design an iterated local search with the ParadisEO-MO packages, you need to chose which ***moAlgo*** in your **moILS**. Then choose the corresponding constructor 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 ***moSolContinue*** 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 ***eoMonOp*** object).

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

Good Luck.

What is next ???

This lesson is maybe difficult but you made it. You can go to the lessons you do not have already followed: **lesson 2** or **lesson 3**.

If you want to do something more challenging, try to find how a ***moAlgo*** can be used in a genetic algorithm (using ParadisEO-EO) as a mutation operator... It is called hybridizing...
