Paradiseo: MoLesson2

Paradiseo-MO Lesson 2: Implementing a Tabu Search

In this lesson, the source code is in *tutorial/Lesson2* and the executable in *build/tutorial/Lesson2*.

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

Open a terminal and from the paradiseo-mo directory, go where the executable of the lesson 2 is (*build/tutorial/Lesson2*).

A tabu search 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 :

./tabu_search ../examples/tsp/benchs/eil101.tsp

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

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

[From] -3347 101 84 20 57 71 92 19 59 10 30 55 27 63 36 26 18 21 46 72 96 35 39 24 29 86 60 40 34 48 67 38 13 0 5 11 16 83 49 50 6 7 87 1 53 90 74 52 58 15 8 95 25 68 61 70 65 45 33 79 23 62 4 37 91 85 78 89 47 69 28 32 66 80 75 94 41 100 44 73 22 9 76 64 97 31 88 51 93 43 2 3 14 56 99 12 82 81 42 17 54 98 77

[To] -697 101 84 90 99 36 97 92 60 15 85 37 43 13 41 42 14 56 86 1 40 21 73 72 20 71 74 55 22 66 38 3 24 54 53 23 28 78 80 8 50 32 2 76 67 79 11 75 49 0 68 26 100 52 27 25 39 57 12 93 94 96 91 58 98 95 5 88 51 17 82 59 4 83 16 44 7 45 46 35 48 63 18 47 81 6 87 61 10 62 89 31 9 30 69 29 19 65 64 70 34 33 77

You have launched a tabu search on the eil101 TSP instance.

A Tabu Search for the TSP

Format of the 'tabu_search.cpp' file:

Now open the 'tabu_search.cpp' file (in *tutorial/Lesson2*). 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 23): it indicates which files are needed to be included in 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 25 to the end) : it contains the tabu search 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.

moTS class:

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

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

You have now the description of the constructor and its 7 parameters. The first six parameters are paradiseo-mo objects (beginning by 'mo') and the seventh 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 moTS 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' that is in *tutorial/example/tsp/*).

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 moTS constructor.

Constructor parameters:

As you have already noticed, 5 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.

- ***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.

- ***moTabuList*** : this object describes the tabu list used by the tabu search. It allows to add a move in the list, to know if a move is in the list and to update the all that the list contains. In our example, the TwoOptTabuList is a moTabuList for the TwoOpt move ('...public moTabuList <TwoOpt>' in two_opt_tabu_list.h). The file two_opt_tabu_list.cpp describes the tabu list in the case of a TwoOpt move.

- ***moAspirCrit***
  : this object corresponds to the aspiration criterion that allows to use a tabu move. In our example, aspir_crit is a template class instanciated with a TwoOpt move ('...moNoAspirCrit <TwoOpt> aspir_crit...' in tabu_search.cpp). This template does not allow to use a tabu move (no aspiration criterion).

- ***moSolContinue***
  : this object corresponds to the stop criterion during the neighborhood inspection. In our example, cont is a template class instanciated with a TwoOpt
  move ('...moGenSolContinue <Route> cont (50000)...' in tabu_search.cpp). This template allows the inspection of the current solution neighbour until a maximum number of steps (in our example 50000).

- ***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').

Instantiate our tabu 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 hill climbing on it (lines 36 to 39):

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 (lines 41 and 42):

RouteEval full_eval ;

full_eval (route) ;

The remaining objects are prepared (lines 44 to 61):

TwoOptInit two_opt_init ;

TwoOptNext two_opt_next ;

TwoOptIncrEval two_opt_incr_eval ;

TwoOptTabuList tabu_list ;

moNoAspirCrit <TwoOpt> aspir_crit ;

moGenSolContinue <Route> cont (10000) ;

The moTS is finally instantiated (line 63):

moTS <TwoOpt> tabu_search (two_opt_init, two_opt_next, two_opt_incr_eval, tabu_list, aspir_crit, cont, full_eval) ;

It can be launched on our initial solution (line 64):

tabu_search (route) ;

Modify the current tabu search:

An example of easy modification that can be possible for this application is to change the number of steps during which a move is considered as a tabu move in the tabu list. In two_opt_tabu_list.cpp the value TABU_LENGTH is defined to be equal to 10, put another positive value (5 for example).

Save your modification ant type 'make' at the prompt of the Lesson2 directory (*build/tutorial/Lesson2*).

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

./tabu_search ../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 tabu search 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 moTS 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 'moNoAspirCrit' object). 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, moIncrEval and moTabuList objects).

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

Good Luck.

What is next ???

You want to try another solution based heuristic, try the lesson 1 to learn to design a ***hill climbing*** algorithm or the lesson 3 to learn to design a ***simulated annealing***.
