Meta-heuristics

Implementing 3 algorithms to solve constraint-satisfaction problems

Aurélien Bernier Levalois & Yoann Fleytoux

aurelien.vincotte@gmail.com/yoann.fleytoux@gmail.com

03.07.2017

Complements of Operational Research

INTRODUCTION

This lab project will consist in the implementations of 3 meta-heuristic algorithms :

- Genetic algorithm
- Tabu search
- Simulated annealing

These 3 algorithms will be solving a general assignment problem with four different set of parameters and a non-linear programming problem available in the lab's description.

All our codes can be found at that link:
https://drive.google.com/open?id=0B5nhnS4CbTWeeVNXX3NLdGxINDQ

We chose to implement the algorithm in python3, being a high level language it is not the best performance wise but it has the advantage to allow quick development.

TABLE OF CONTENTS

DESCRIPTION OF THE PROBLEMS

Problem 1 : Generalized assignment problem

The generalized assignment problem is a combinatorial optimization problem where m agents have to realize together n tasks, each task having a different cost and resource usage depending on what agent is realizing it. Each agent also has a different maximum resource capacity. The goal is to minimize the total cost.

The 4 problems that will be solved are defined in the given file PAG2017.txt, in the following format :

Number of agents (m), number of tasks (n)

m*n matrix of the costs of each task for each agent

m*n matrix of the resources used by each agent while realizing each task

Maximum capacity of each agent

A parser has been realized in the file myParser.py which allows to get all the data in a comprehensive way from the text file (we changed PAG2017.txt, there were some spaces/tabulation weirdly place, besides that the data were not touch)

Problem 2 : Non-linear programming problem

The non-linear programming problem is defined by :

$$
\begin{array}{ll}
\text{min} & f(x) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141 \\
s.\ to & g_1(x) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \le 0 \\
 & g_2(x) = -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \le 0 \\
 & g_3(x) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \le 0 \\
 & g_4(x) = -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \le 0 \\
 & g_5(x) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \le 0 \\
 & g_6(x) = -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \le 0 \\
 & 78 \le x_1 \le 102 \\
 & 33 \le x_2 \le 45 \\
 & 27 \le x_i \le 45, \qquad i = 3,4,5.
\end{array}
$$

DESCRIPTION OF THE META-HEURISTIC METHODS

Algorithm 1 : Genetic algorithm

The genetic algorithm is inspired by the process of natural selection. It’s an evolutionary algorithm. It works by allowing to crossover solutions as if they were composed of genes, and then allowing random mutations before choosing the best solution.

Algorithm 2 : Tabu search

The tabu search meta-heuristic is a very common method to solve many operational research problems. It is a local search method with added memory. The difference is that the tabu-search allows to advance to moves that give a worst score than the current solution, while restraining the algorithm from going back to recent solutions.

Algorithm 3 : Simulated annealing

Simulated annealing is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time, simulated annealing may be preferable to alternatives such as gradient descent.

IMPLEMENTATIONS:

Solving problem 1 : Generating first random solutions

In both approach, random appropriate solutions needed to be generated. To generate a solution our method was:

While all the tasks were not assigned

starting at a random agent

starting at a random task

try to assigned the task to the agent randomly (we had good results with a 0.5 probability), if the probability is passed, remove the last task assigned to the agent until he can be assigned the new task. If not passed, try with the next agent. When the task is assigned, we pass at the next task.

Since the constraints are hard to satisfy, especially for the problems with the biggest set of data (problem 3-4), after a set number of tries, the solution is discarded and the process is started again. That way we had the best results to generate solutions quickly.

Here a few results for each problem:

Problem numero:	Number Of solution Generated:	Time in seconds:	Average solution/second:
1	100	9.91	0.0991
2	100	5.85	0.0585
3	100	109.33	1.0933
4*	25	161.05	6.442

*the 4th problem have the hardest random solution to find: for example in the above sample a solution can be generated between 0.65 and 39.24 seconds if unlucky

Solving problem 1 : Genetic Algorithm

After the generation of a few random solution (n), the algorithm select randomly couples (number of couples <= n!/(2!*(n-2)!), the fittests have higher chance of being chosen) and the algorithm generate a new solution based on their answer and random mutations. Couples who have already mated or which are selected with themselves are discarded until we have enough couples.

To generate the new solution, until all the task are assigned we try to assign the task the an agent if: a probability check is passed (we had good results with a probability between 0.2 and 0.1) OR if the agent has the task assigned in either of his parents. Like for the random generations, task are removed to fit the new task if the agent doesn't have the right capacity. Passing the probability check is a way to introduce random Mutations in the population.

Here are a few results for each problem (the stopping criteria is a set runtime after the original population generation, more details, like the proportion of mutated genes, can be found at the link provided, look for try results):

Problem 1: (sizePopulationStart=30 ,sizePopulation=80, probaMutGate=0.2)

Try number	Total runtime	Number of solutions generated	Best solution found
1	32.67	6510	**262**
2	33.59	7070	**272**
3	32.41	6590	**264**
4	33.43	6590	**264**
5	34.68	6510	**261**

Problem 2: (sizePopulationStart=20 ,sizePopulation=50, probaMutGate=0.2)

Try number	Total runtime	Number of solutions generated	Best solution found
1	41.1	4270	**435**
2	41.52	3920	**436**
3	41.43	4120	**435**

4	41.31	3870	**438**
5	41.94	4120	**430**

Problem 3: (sizePopulationStart=20 ,sizePopulation=30, probaMutGate=0.1)

Try number	Total runtime	Number of solutions generated	Best solution found
1	74.61	410	**742**
2	72.78	790	**757**
3	79.79	940	**737**
4	69.47	790	**758**
5	70.63	1000	749

Problem 4: (sizePopulationStart=5 ,sizePopulation=10, probaMutGate=0.02)

Try number	Total runtime	Number of solutions generated	Best solution found
1	159.81	435	**1165**
2	242.95	190	**1152**
3	251.23	70	**1176**
4	254.61	530	**1181**
5	221.66	310	**1155**

Conclusion:

The genetic approach functions well for the first and second problem, but as the complexity augment, we take more time to generate a population, so we have less genetic diversity, so we test less solutions, and we struggle to obtain good results (and the algorithm can get stuck during the mating process).

Solving problem 1 : Tabu Search

Our implementation of the tabu search generates first solutions the same way the genetic algorithm does.

We then use the best solution as the base, and start making random swaps of tasks in between agents, while putting the last configurations in the tabu.

A considerable part of the algorithm is finding swapping possibilities between two tasks, given the number of agents and tasks in harsher problems.

This way, the algorithm won't get stuck in local minimals because we can actually go through new solutions that are worse, as long as they are not in the list of tabus.

Problem 1: 10000 iterations of tabu search - best solution : 261 in 47 seconds

261:

```
Agent  0 :8  14
Agent  1 :1  10  13
Agent  2 :2  5  7  12
Agent  3 :4  6  9
Agent  4 :0  3  11
```

Problem 2: 10000 iterations of tabu search - best solution : 427 in 72 seconds

Problem 3: 10000 iterations of tabu search - best solution : 734 in 136 seconds

Problem 4: 10000 iterations of tabu search - best solution : 1120 in 228 seconds

Each problem has been solved once.

We finish with decent results, compared to the genetic algorithm, we get a better result in the 2nd, the 3rd and the 4th and we obtain the same result in the 1st.

Solving problem 2 : Simulated Annealing

After generating randomly solutions, we try to move each parameters randomly (one at the time) in a close neighborhood. At first the temperature is pretty high and the algorithm function similarly to “random walk”, as the temperature gets lowered the algorithm adopt a fonctionnement closer to “hill climbing”. When the temperature is at the lowest, we start from a new point.

More results can be found at the link provided

Problem2(Starting Temperature=2000, step size=0.1, NeighborhoodSize=1):

```
nb Random Restart: 9
nb Iteration: 143352
[78.14, 33.2, 30.15, 44.92, 36.46]
Score: -30626.7716213592
Total execution time: 60 seconds
```

```
nb Random Restart: 8
nb Iteration: 1242594
[78.06, 33.26, 30.19, 44.81, 36.36]
Score: -30625.7842894568
Total execution time: 60
```

```
nb Random Restart: 9
nb Iteration: 1454642
[78.01, 33.31, 30.2, 44.94, 36.31]
Score: -30629.1922310718
Total execution time: 60
```

Conclusion:

We don’t have other method to compare our results, but it seems to be working well.

Solving problem 2 : Tabu search

We didn’t have the time to finish to adapt the tabu search for the second problem, but we think that the tabu approach would yield interesting results with this kind of problem.

CONCLUSION

To conclude, we have used our 3 implementations to try and solve the general affectation problem and the non-linear programming problem given.

We can see that the results of the tabu search are slightly more satisfying for the general affectation problem than the genetic algorithm that gets slightly similar (which could be solved by a more thorough implementation and better heuristics). We can however see that the comparison is not entirely fair as the tabu search takes a bit longer with the number of iterations we chose (10 000). We would admit that the implementations could use better heuristics to make them stronger/faster (for example the tabu search would be better with intensification and diversification).

As for the second problem, the simulated annealing gives good results in an acceptable length of time, however we don’t have another algorithm to compare how efficient it is.