**python DEAP學習3（遺傳演算法） 0-1背包問題**

翻譯 2017年05月27日 14:47:53

* [python](http://so.csdn.net/so/search/s.do?q=python&t=blog) /
* [遺傳演算法](http://so.csdn.net/so/search/s.do?q=%E9%81%97%E4%BC%A0%E7%AE%97%E6%B3%95&t=blog) /

Knapsack Problem: Inheriting from Set

Again for this example we will use a very simple problem, the 0-1 Knapsack. The purpose of this example is to show the simplicity of DEAP and the ease to inherit from anything else than a simple list or array.

Many evolutionary algorithm textbooks mention that the best way to have an efficient algorithm is to have a representation close the problem. Here, what can be closer to a bag than a set? Lets make our individuals inherit from the [set](https://docs.python.org/2/library/stdtypes.html#set) class.

**[python]** [view plain](https://blog.csdn.net/forethougtht/article/details/72781357) [copy](https://blog.csdn.net/forethougtht/article/details/72781357)

1. creator.create("Fitness", base.Fitness, weights=(-1.0, 1.0))#最小化第一個參數背包重量，最大化第二個參數背包價值
2. creator.create("Individual", set, fitness=creator.Fitness)

用set（）函數創建個體...具體怎麼用待查。

That’s it. You now have individuals that are, in fact sets, they have the usual attribute fitness. The fitness is a minimization of the first objective (the weight of the bag) and a maximization of the second objective (the value of the bag). We will now create a dictionary of 100 random items to map the values and weights.

**[python]** [view plain](https://blog.csdn.net/forethougtht/article/details/72781357) [copy](https://blog.csdn.net/forethougtht/article/details/72781357)

1. # Create the item dictionary: item name is an integer, and value is
2. # a (weight, value) 2-uple.
3. items = {}
4. # Create random items and store them in the items' dictionary.
5. **for** i **in** range(NBR\_ITEMS):
6. items[i] = (random.randint(1, 10), random.uniform(0, 100))

用隨機函數創建背包，重量為[1,10]，價值為[0,100]。

We now need to initialize a population and the individuals therein. For this, we will need a [Toolbox](http://deap.readthedocs.io/en/master/api/base.html#deap.base.Toolbox) to register our generators since sets can also be created with an input iterable.

**[python]** [view plain](https://blog.csdn.net/forethougtht/article/details/72781357) [copy](https://blog.csdn.net/forethougtht/article/details/72781357)

1. toolbox = base.Toolbox()
2. toolbox.register("attr\_item", random.randrange, NBR\_ITEMS)
3. toolbox.register("individual", tools.initRepeat, creator.Individual,
4. toolbox.attr\_item, IND\_INIT\_SIZE)
5. toolbox.register("population", tools.initRepeat, list, toolbox.individual)

利用toolbox.register初始化種群和個體，用法與前一篇文章相同。

Voilà! The last thing to do is to define our evaluation function.

最後定義進化函數。

**[python]** [view plain](https://blog.csdn.net/forethougtht/article/details/72781357) [copy](https://blog.csdn.net/forethougtht/article/details/72781357)

1. **def** evalKnapsack(individual):
2. weight = 0.0
3. value = 0.0
4. **for** item **in** individual:
5. weight += items[item][0]
6. value += items[item][1]
7. **if** len(individual) > MAX\_ITEM **or** weight > MAX\_WEIGHT:
8. **return** 10000, 0             # Ensure overweighted bags are dominated
9. **return** weight, value

Everything is ready for evolution. Ho no wait, since DEAP’s developers are lazy, there is no crossover and mutation operators that can be applied directly on sets. Lets define some. For example, a crossover, producing two children from two parents, could be that the first child is the intersection of the two sets and the second child their absolute difference.

萬事俱備，只差操作運算元了，首先是選擇....這段代碼沒有看懂....

**[python]** [view plain](https://blog.csdn.net/forethougtht/article/details/72781357) [copy](https://blog.csdn.net/forethougtht/article/details/72781357)

1. **def** cxSet(ind1, ind2):
2. """Apply a crossover operation on input sets. The first child is the
3. intersection of the two sets, the second child is the difference of the
4. two sets.
5. """
6. temp = set(ind1)                # Used in order to keep type
7. ind1 &= ind2                    # Intersection (inplace)
8. ind2 ^= temp                    # Symmetric Difference (inplace)
9. **return** ind1, ind2

A mutation operator could randomly add or remove an element from the set input individual.

變異，隨機挑選個體淘汰，或者加入一些隨機生成的個體。

**[python]** [view plain](https://blog.csdn.net/forethougtht/article/details/72781357) [copy](https://blog.csdn.net/forethougtht/article/details/72781357)

1. **def** mutSet(individual):
2. """Mutation that pops or add an element."""
3. **if** random.random() < 0.5:
4. **if** len(individual) > 0:     # We cannot pop from an empty set
5. individual.remove(random.choice(sorted(tuple(individual))))
6. **else**:
7. individual.add(random.randrange(NBR\_ITEMS))
8. **return** individual,

We then register these operators in the toolbox. Since it is a multi-objective problem, we have selected the SPEA-II selection scheme : [selSPEA2()](http://deap.readthedocs.io/en/master/api/tools.html#deap.tools.selSPEA2)

然後我們在toolbox裡面註冊操作運算元，因為這是一個多目標的問題，我們選擇SPEA-II方法....具體方法還要瞭解一下....為什麼前面自己定義了選擇和變異，還要用toolbox裡面的選擇、變異函數...不懂。

**[python]** [view plain](https://blog.csdn.net/forethougtht/article/details/72781357) [copy](https://blog.csdn.net/forethougtht/article/details/72781357)

1. toolbox.register("evaluate", evalKnapsack)
2. toolbox.register("mate", cxSet)
3. toolbox.register("mutate", mutSet)
4. toolbox.register("select", tools.selNSGA2)

From here, all that is left to do is either write the algorithm or use provided in [algorithms](http://deap.readthedocs.io/en/master/api/algo.html#module-deap.algorithms). Here we will use the [eaMuPlusLambda()](http://deap.readthedocs.io/en/master/api/algo.html#deap.algorithms.eaMuPlusLambda) algorithm.

**[python]** [view plain](https://blog.csdn.net/forethougtht/article/details/72781357) [copy](https://blog.csdn.net/forethougtht/article/details/72781357)

1. **def** main():
2. NGEN = 50
3. MU = 50
4. LAMBDA = 100
5. CXPB = 0.7
6. MUTPB = 0.2
8. pop = toolbox.population(n=MU)
9. hof = tools.ParetoFront()
10. stats = tools.Statistics(**lambda** ind: ind.fitness.values)
11. stats.register("avg", numpy.mean, axis=0)
12. stats.register("std", numpy.std, axis=0)
13. stats.register("min", numpy.min, axis=0)
14. stats.register("max", numpy.max, axis=0)
16. algorithms.eaMuPlusLambda(pop, toolbox, MU, LAMBDA, CXPB, MUTPB, NGEN, stats,
17. halloffame=hof)
19. **return** pop, stats, hof

Finally, a [ParetoFront](http://deap.readthedocs.io/en/master/api/tools.html#deap.tools.ParetoFront) may be used to retrieve the best non dominated individuals of the evolution and a [Statistics](http://deap.readthedocs.io/en/master/api/tools.html#deap.tools.Statistics) object is created for compiling four different statistics over the generations. The Numpy functions are registered in the statistics object with the axis=0 argument to compute the statistics on each objective independently. Otherwise, Numpy would compute a single mean for both objectives.

The complete [examples/ga/knapsack](https://github.com/DEAP/deap/blob/08986fc3848144903048c722564b7b1d92db33a1/examples/ga/knapsack.py).

**完整代碼：**

**[python]** [view plain](https://blog.csdn.net/forethougtht/article/details/72781357) [copy](https://blog.csdn.net/forethougtht/article/details/72781357)

1. #    This file is part of DEAP.
2. #
3. #    DEAP is free software: you can redistribute it and/or modify
4. #    it under the terms of the GNU Lesser General Public License as
5. #    published by the Free Software Foundation, either version 3 of
6. #    the License, or (at your option) any later version.
7. #
8. #    DEAP is distributed in the hope that it will be useful,
9. #    but WITHOUT ANY WARRANTY; without even the implied warranty of
10. #    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
11. #    GNU Lesser General Public License for more details.
12. #
13. #    You should have received a copy of the GNU Lesser General Public
14. #    License along with DEAP. If not, see <http://www.gnu.org/licenses/>.
16. **import** random
18. **import** numpy
20. **from** deap **import** algorithms
21. **from** deap **import** base
22. **from** deap **import** creator
23. **from** deap **import** tools
25. IND\_INIT\_SIZE = 5
26. MAX\_ITEM = 50
27. MAX\_WEIGHT = 50
28. NBR\_ITEMS = 20
30. # To assure reproductibility, the RNG seed is set prior to the items
31. # dict initialization. It is also seeded in main().
32. random.seed(64)
34. # Create the item dictionary: item name is an integer, and value is
35. # a (weight, value) 2-uple.
36. items = {}
37. # Create random items and store them in the items' dictionary.
38. **for** i **in** range(NBR\_ITEMS):
39. items[i] = (random.randint(1, 10), random.uniform(0, 100))
41. creator.create("Fitness", base.Fitness, weights=(-1.0, 1.0))
42. creator.create("Individual", set, fitness=creator.Fitness)
44. toolbox = base.Toolbox()
46. # Attribute generator
47. toolbox.register("attr\_item", random.randrange, NBR\_ITEMS)
49. # Structure initializers
50. toolbox.register("individual", tools.initRepeat, creator.Individual,
51. toolbox.attr\_item, IND\_INIT\_SIZE)
52. toolbox.register("population", tools.initRepeat, list, toolbox.individual)
54. **def** evalKnapsack(individual):
55. weight = 0.0
56. value = 0.0
57. **for** item **in** individual:
58. weight += items[item][0]
59. value += items[item][1]
60. **if** len(individual) > MAX\_ITEM **or** weight > MAX\_WEIGHT:
61. **return** 10000, 0             # Ensure overweighted bags are dominated
62. **return** weight, value
64. **def** cxSet(ind1, ind2):
65. """Apply a crossover operation on input sets. The first child is the
66. intersection of the two sets, the second child is the difference of the
67. two sets.
68. """
69. temp = set(ind1)                # Used in order to keep type
70. ind1 &= ind2                    # Intersection (inplace)
71. ind2 ^= temp                    # Symmetric Difference (inplace)
72. **return** ind1, ind2
74. **def** mutSet(individual):
75. """Mutation that pops or add an element."""
76. **if** random.random() < 0.5:
77. **if** len(individual) > 0:     # We cannot pop from an empty set
78. individual.remove(random.choice(sorted(tuple(individual))))
79. **else**:
80. individual.add(random.randrange(NBR\_ITEMS))
81. **return** individual,
83. toolbox.register("evaluate", evalKnapsack)
84. toolbox.register("mate", cxSet)
85. toolbox.register("mutate", mutSet)
86. toolbox.register("select", tools.selNSGA2)
88. **def** main():
89. random.seed(64)
90. NGEN = 50
91. MU = 50
92. LAMBDA = 100
93. CXPB = 0.7
94. MUTPB = 0.2
96. pop = toolbox.population(n=MU)
97. hof = tools.ParetoFront()
98. stats = tools.Statistics(**lambda** ind: ind.fitness.values)
99. stats.register("avg", numpy.mean, axis=0)
100. stats.register("std", numpy.std, axis=0)
101. stats.register("min", numpy.min, axis=0)
102. stats.register("max", numpy.max, axis=0)
104. algorithms.eaMuPlusLambda(pop, toolbox, MU, LAMBDA, CXPB, MUTPB, NGEN, stats,
105. halloffame=hof)
107. **return** pop, stats, hof
109. **if** \_\_name\_\_ == "\_\_main\_\_":
110. main()