Module for Montage Lines

PyBalance is a Python package made for use as a guide in the creation, processing and analysis of assembly lines.

PyBalance provides:

- Tools that assist in the creation, balancing and analyzing of assembly lines.
- A standart programming interface that is suitable for many applications
- Ability to work with large montage lines
- Inspecting assembly lines, performance analysis, calculation of smoothness index, line efficiency, balance delay, minimum cycle time, productivity

With pybalance, you can get detailed information about assembly lines, change their station sequences, find alternative solutions, and more.

Pybalance was born in October 2020. First version was designed and written by Onur M. Çeldir.

Tutorial

This guide can help you start working with PyBalance.

Starting to Pybalance

This section describes the most basic function of pybalance for beginners.

Creating an empty montage line.

```
>>> import pybalance as pb
>>> line = pb.Line()
```

A line contains stations and tasks. Each task carries the task number, predecessors and duration of the task in it `[task_number, [predecessors], task_time]`. Tasks are defined to fill this line.

```
>>> line.add_task([
>>>    [1, [0], 2],
>>>    [2, [1], 5],
>>>    [3, [1], 4],
>>>    [4, [2,3], 3]
>>> ])
```

You can balance the line you have created. Pybalance can balance many line types. If no parameters are entered, uses heuristic algorithms for single pattern lines. For this, `cycle_time` must be entered as a parameter.

```
>>> output = line.balance(6) # 6 is cycle time
>>> output
[[1, 3], [2], [4]]
```

This result shows us the stations of montage line. Each element of the array is a separate station.

Line Balancing

Pybalance contains many balancing algorithms. The simplest to use is the `balance(cycle_time)` function. The following functions can be applied for more detailed operations.

Straight Line Balancing

Heuristic, mathematical and experimental methods can be used in balancing straight assembly lines.

Heuristic methods:

Heuristic methods are commonly used to balance these lines. It is sufficient to enter `cycle_time` and `method` type (largest candidate rule(`'lcr'`) or helgeso-birnie(`'hb'`)) in heuristic methods. If method name is not entered, largest candidate rule(`'lcr'`) is used by default. Heuristic methods always give the same result.

```
>>> montage_line_1 = line.heuristic_method(12) # basic usage
>>> montage_line_1
[[1, 2, 3, 6], [4, 5, 7], [8, 9]]

>>> montage_line_2 = line.heuristic_method(cycle_time=12, method="hb") # with parameters
>>> montage_line_2
[[1, 3, 2], [5, 4, 6, 7], [8, 9]]
```

COMSOAL algorithm:

COMSOAL is an algorithm where probabilities are generated randomly. Basic usage of COMSOAL is like this.

```
>>> result = line.comsoal_method(12) # basic usage, returns best result(cycle_time=12)
[[1, 2, 5], [3, 4, 6], [7, 8, 9]]
```

It makes intuitive task assignment if local search parameter is not entered. Desired amount of iteration is done and desired amount of result is obtained. If desired, the best result can be brought. The `out` parameter specifies how many desired results will be returned. If it is called `out='max'` it brings the all results. If a number is entered, it returns the given number of results. Default `iteration` is 100. `local_search` parameter will be explain detailed in the next sections.

```
>>> # usage with parameters, returns best result
>>> # in this result we have 4 results. Because out parameter is 4.
>>> line.comsoal_method(
>>>     cycle_time=12, # cycle time, default value is auto calculated.
>>>     iteration=250, # how much iteration? Default value is 100
>>>     local_search="heuristic", # selecting local search method, default is "local"
>>>     out=4 # how much result wanted? default value is 1.
>>> )
 [[[1, 2, 5], [3, 4, 6], [7, 8, 9]],
  [[1, 3, 2], [5, 6, 4, 7], [8, 9]],
  [[1, 2, 3, 6], [5, 4, 7], [8, 9]],
  [[1, 3, 2, 6], [4, 5, 7], [8, 9]]]
```

Genetic Algorithms:

The use of genetic algorithms can increase the time it takes to get results. But probability of finding better results is higher. It needs only `cycle_time` parameter by default. Probability mutation(`p_m`), probability crossover(`p_c`), number of generations(`generation`), population size(`size`), local search(`local_search`), desired number of outputs(`out`) are optional.

Warning! Changing with these parameters can dramatically alter processing times and results:

```
>>> result = line.genetic_algorithms(cycle_time=12, local_search='genetics', p_c=0.5)
>>> result
[[1, 3, 6], [2, 5, 4], [7, 8, 9]]
```

Type-U Line Balancing

Pybalance contains Type-U montage line solutions. As in other algorithms, if only `cycle_time` information is entered as parameter, it will show the fastest solution.

Heuristic algorithms and COMSOAL algorithm can be used to solve these montage lines. Heuristic algorithms are recommended by default(`method='lcr'`, it can be `'comsoal'` or `'hb'`).

Output returns 2 results. The first of these is the task sequence, the second is the information whether the line is in the front and back(F: front, B: back, F-B: front or back).

```
>>> result = line.u_type_balance(12)
>>> result[0] # returns stations and tasks
[[1, 9, 8, 2], [7, 3, 4], [5, 6]]
>>> result[1] # returns task side: front(F), back(B), front or back(F-B)
['F', 'B', 'B', 'F', 'B', 'F', 'F-B', 'F-B', 'F-B']
```

More detailed searches can be made with different parameters if desired. `out` parameter determines the amount of results we will get.

```
>>> result = line.u_type_balance(cycle_time=11, method='comsoal', iteration=100, out=1)
>>> result[0]
[[1, 9, 8], [7, 2, 5], [3, 4, 6]]
>>> result[1]
['F', 'B', 'B', 'B', 'F', 'F-B', 'F', 'F-B', 'F-B']
```

Local Search Procedure

Local search allows us to find different solutions in the same task sequence. It is necessary to give station sequence as a parameter.

```
>>> initial_solution = [[1, 3, 2, 6], [5, 4, 7], [8, 9]]
>>> ls_solution = line.local_search_procedure(initial_solution, cycle_time=12)
>>> ls_solution
[[1, 3, 2], [6, 5, 4, 7], [8, 9]]
```

Heuristic methods (`'heuristic'`), predictive algorithms(`'local'`) or genetic algorithms(`'genetics'`) can be used. Genetic algorithms gives us the highest line efficiency but it uses much more calculation times. `out` parameter gives desired number of outputs.

```
>>> initial_solution = [[1, 3, 2, 6], [5, 4, 7], [8, 9]]
>>> ls_solution = line.local_search_procedure(
>>>    initial_solution,
>>>    cycle_time=12,
>>>    local_search='local', # it should be 'genetics' or 'heuristic'
>>>    out=4
>>> )
>>> ls_solution # returns 4(out) different solutions.
[[[1, 3, 2, 6], [5, 4, 7], [8, 9]],
 [[1, 3, 2, 6], [5, 4], [7, 8, 9]],
 [[1, 3, 2], [6, 5, 4, 7], [8, 9]],
 [[1, 3, 2], [6, 5, 4], [7, 8, 9]]]
```

Analyzing Montage Lines

With pybalance, we can get detailed information about the assembly lines we use. We can change the necessary information and re-balance the montage line.

Smooth Index

Measure of how much balanced the line is. It needs only `station_list` as a parameter. `cycle_time` will auto calculated but it can be given as a parameter.

```
>>> my_line = [[1, 3, 2, 6], [5, 4, 7], [8, 9]]
>>> line.calculate_smooth_index(my_line, cycle_time=12)
5.385164807134504 # %
```

Line Efficiency

Line efficiency means the efficiency of working times as a result of balancing in assembly line.

```
>>> my_line = [[1, 3, 2, 6], [5, 4, 7], [8, 9]]
>>> line.calculate_line_efficiency(my_line, cycle_time=12)
85.04120886907083 # %
```

Loss of Balance

Means the ratio between work station leisure time and cycle time.

```
>>> my_line = [[1, 3, 2, 6], [5, 4, 7], [8, 9]]
>>> line.calculate_loss_balance(my_line)
14.958791130929171 # %
```

Total Work Time and simple Task Time

Total Work Time: Sum of the work time in assembly line.

Task Time: Time of the simple task

Station Time: Sum of the work time in station

```
>>> my_line = [[1, 3, 2, 6], [5, 4, 7], [8, 9]]
>>> line.total_work_time(my_line)
29
>>> line.get_task_time(1)
2
>>> line.get_station_time(my_line[0])
12
```

Learn By Doing Example

Tell me, I'll forget;

Show me, I may remember;

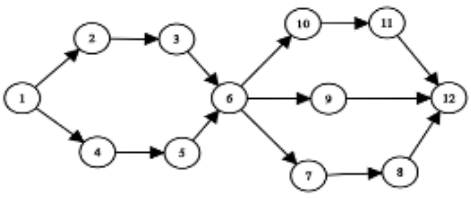
But involve me, I will learn.

- Benjamin Franklin -

Questions

1. Import pybalance and create empty montage line.

2. Enter the information of the montage line given the task diagram and times.



Task	Predecessors	Task Time
1	-	5
2	1	3
3	2	4
4	1	3
5	4	6
6	3,5	5
7	6	2
8	7	6
9	6	1
10	6	4
11	10	4
12	8, 9, 11	7

3. Balance the montage line using heuristic methods. (cycle time=18)

4. Balance the montage line using Comsoal algorithm and fetch the best 3 solutions (cycle_time=20) and print their line efficiencies.

5. Rearrange the `[[1, 4, 2, 3], [5, 6, 7, 8, 9], [10, 11, 12]]` using local search procedure and print its line efficiency(Old and new).

6. Using answer of Question 5 calculate station time of station 2.

Answers

1.
```
>>> import pybalance as pb
>>> line = pb.Line()
```

2.
```
>>> line.add_task[[1, [0], 5],
>>>               [2, [1], 3],
>>>               [3, [2], 4],
>>>               [4, [1], 3],
>>>               [5, [4], 6],
>>>               [6, [3,5], 5],
>>>               [7, [6], 2],
>>>               [8, [7], 6],
>>>               [9, [6], 1],
>>>               [10, [6], 4],
>>>               [11, [10], 4],
>>>               [12, [8,9,11], 7]])
```

3.
```
>>> line.heuristic_method(18)
[[1, 4, 2, 5], [3, 6, 7, 10, 9], [8, 11, 12]]
```

4.
```
>>> solutions = line.comsoal_algorithm(cycle_time=20, out=3)
>>> for solution in solutions:
>>>    print(solution, 'Line Efficiency:', line.calculate_line_efficiency(solution))
[[1, 4, 2, 5], [3, 6, 7, 8], [10, 11, 9, 12]] Line Efficiency: 98.03921568627452
[[1, 2, 3, 4], [5, 6, 7, 10], [8, 11, 9, 12]] Line Efficiency: 94.14393025894745
[[1, 2, 3, 4], [5, 6, 9, 10, 7], [8, 11, 12]] Line Efficiency: 94.14393025894745
```

5.
```
>>> old_line = [[1, 4, 2, 3], [5, 6, 7, 8, 9], [10, 11, 12]]
>>> new_line = line.local_search_procedure(old_line, cycle_time=20)
>>> print(old_line, ':', line.calculate_line_efficiency(old_line))
[[1, 4, 2, 3], [5, 6, 7, 8, 9], [10, 11, 12]] : 88.21488698022421
>>> print(new_line, 'Line Efficiency:', line.calculate_line_efficiency(new_line))
[[1, 4, 2, 3], [5, 6, 7, 8], [9, 10, 11, 12]] : 91.2280701754386
```

6.
```
>>> line.get_station_time(new_line[1])
16
```

Author: Onur Mert ÇELDİR

Date: 2020, October