Optimization Algorithms Library (OAL)

Software Requirements Specification

Version 0.0.1 approved on June 27, 2021.

Established by Mohammed Alqmase
Email: alqumasi@gmail.com

Abstract

Optimization-algorithms is a Python library that contains useful algorithms for several complex problems such as partitioning, floor planning, scheduling. This library will provide many implementations for many optimization algorithms. This library is organized in a problem-wise structure. For example, there are many problems such as graph partitioning problem, scheduling problem, etc. For each problem, there are many algorithms to solve it. For each algorithm, there are many possible approaches to implement it.

Q1: what is the goal of this library?

The goal is to provide a python library **for the purpose of** optimization algorithms **with respect to** consistency and simplicity **from the viewpoint of** researchers, students, and industries **in the context of** VLSI circuit design, parallel computing, cloud computing, and artificial intelligent.

Q2: what is the motivation of building this library?

During my research in partitioning problem, I found that there is a lack of library where the researchers can easily compare their algorithms with others. I found that each implementation has its own input style and output style which means more afford is spent for understanding the codes and adapting the input. We suggest this library where the optimization algorithms can be packaged with consistency in mind.

Q3: who are the users?

1. Students
2. Researchers
3. Industries

Q4: what are the features of this library?

1. ***Consistency:*** we are striving to ensure consistency.
2. ***Simplicity:*** we are striving to ensure simplicity.
3. ***Source for learning, and teaching:*** we are striving to make this library an essential source for learning and teaching optimization algorithms.
4. ***Useful resource for researchers, students, and industries:*** we are striving to make this library an essential resource for researchers, students, and industries.

Q5: what we mean by ensuring the consistency?

Consistency is the main feature of this library. We will strive to be consistent in project structure, coding, documentation, videos, and datasets. The following provide a basic consistency points of this library:

1. Consistency in the project structure
   - Consistency in guidelines, styles, and templates.
   - Consistency in creating new modules and packages.
   - Consistency in the file names, folders names, folders hierarchies, configuration files, files arrangement and distributions.
   - Consistency in dates and times of updating and reviewing.
   - Consistency in working teams and reviews.
   - Consistency in evaluation tests and validation.
   - Consistency in reviews methodology.
   - Consistency in the code
2. Consistency within one module or function
   - Consistency in return statements.
   - Consistency in input, and output.
   - Consistency in print and summary.
   - Consistency in signatures of objects, methods, or data attributes.
   - Consistency in code style. The style of coding should be consistent and following special best practices guides such as spaces, comments, etc.
   - Consistency in the data structure.
3. Consistency in the documentation
   - Consistency in describing problems, algorithms, and implementations.
   - Consistency in describing classes, methods, attributes.
   - Consistency in titles.
   - Consistency in adding new sections and chapters.
   - Consistency in naming conventions. The naming conventions should follow a certain naming standard.
   - Consistency in writing many special names such as the names of operating systems, programming languages, standards bodies, etc.
   - Consistency in writing abbreviations.
   - Consistency in references.
   - Consistency in code documentation.
   - Consistency in describing the datasets.
4. Consistency in the videos
   - Video length should follow ultimate guide to ensure consistency.
   - The methodology of describing problems, algorithms, implementation should ensure consistency.
   - All videos should be hosted in one place such as one YouTube channel.
   - The videos should be categorized properly with consistency in mind.
   - Consistency in ordering and naming
   - Consistency with the documentation
5. Consistency in the datasets

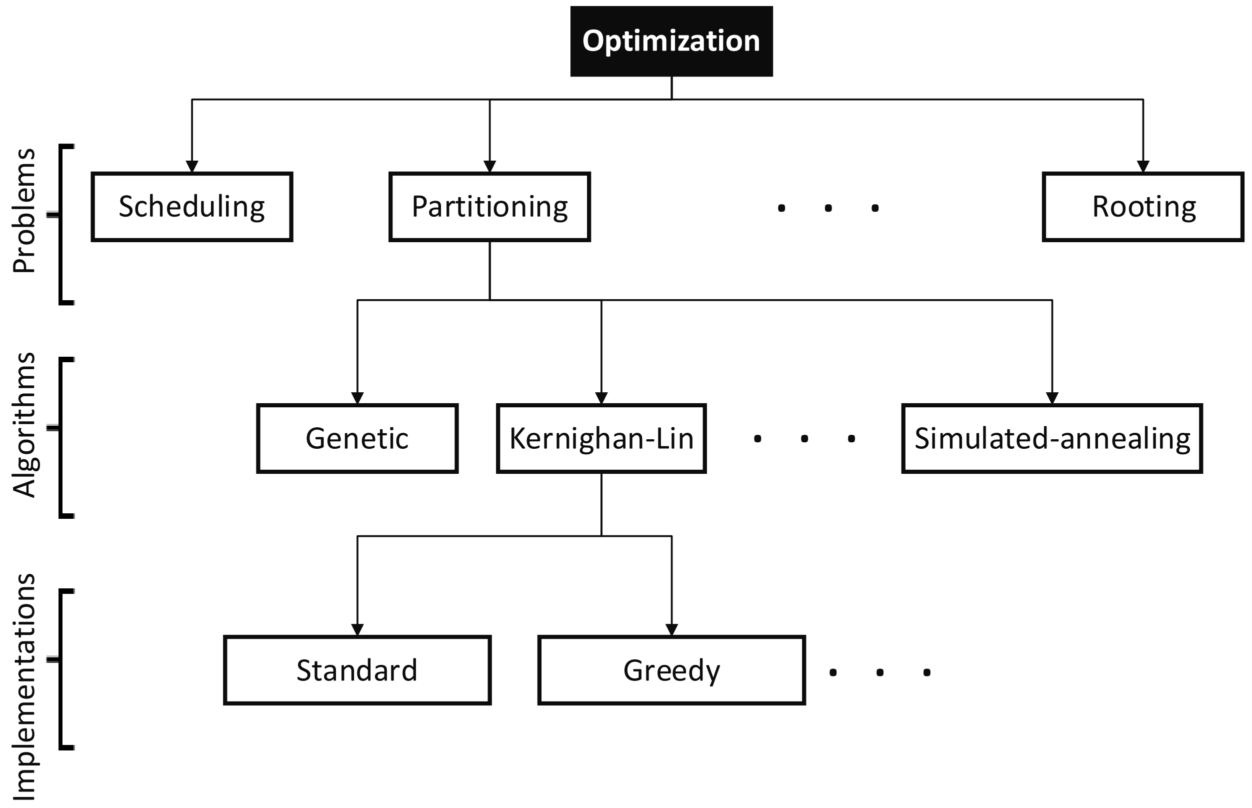
- The structure of the dataset should follow a certain formatting and structure.
- The test datasets should be consistent to validate and ensure the quality of the implemented algorithms.

Q6: what are the metrics that will be used to measure success?

- Number of installations
- Number of problems that are considered.
- Number of algorithms that are considered.
- Number of implementations
- Number of Awards
- Number of citations
- Number of visitors/readers
- Number of developers/contributors
- Number of reviewers
- Number of versions
- Number of issues
- Number of comments
- Number of research groups that use or support -OAL-
- Number of industries that use or support -OAL-
- Number of students

Q7: what is the structure of this library?

- Optimization-Algorithms
    - Partitioning Problem
        - Kernighan-Lin (KL)
        - Simulated Annealing (SA)
        - Hybridization SA&KL (SAKL)
        - Others not Yet implemented.
    - Floor Planning Problem
        - Algorithms not Yet implemented.
    - Scheduling Problem
        - Algorithms not Yet implemented.
    - Rooting Problem
        - Algorithms not Yet implemented.
    - Other problems not Yet defined.



Q8: what is our future plan?

Date	Version	Main Deliverable Features
Dec 30th, 2021	**0.1.0**	• Formulating at least three problems • Implementing at least ten optimization-algorithms • Stablishing official website for this project • Improving the consistency
Dec 30, 2022	**0.2.0**	• Formulating at least three new problems • Implementing at least ten optimization-algorithms • Publishing at least one paper • By this time, the author expects that we receive at least three citations. • By this time, the author expects that the library will be a good resource in industries. • By this time, the author expect that the library tutorial will be a recommended source for learning optimization algorithm. • By this time, the author expect that the library receives a number of contributions from researchers.
Dec 30, 2023	**0.3.0**	• Formulating at least three new problems • Implementing at least ten optimization-algorithms. • Publishing at least one new paper.

		• By this time, the author expects that the library will be cited by at least new thirty citations. • By this time, the author expects that the number of industries that used the library is increased by at least half. • By this time, the author expect that the library tutorial will be a recommended source for learning optimization algorithm. • By this time, the author expect that the library receives a number of contributions from researchers.
Dec 30, 2024	**0.4.0**	• By this time, the author expects that the library becomes essential library for many industries. • By this time, the author expects that the library becomes essential library for many researchers. • By this time, the author expects that the library becomes well-known library in the academic environment.

Q9: what is the development methodology?
The author adapts the GitHub methodology for contributing to this library. The proposed methodology has two steps. The first step is formulating the problem. This library is problem-wise library where the problem should be formulated into inputs and outputs with consistency in mind. The second step is contributing to this problem by implementing optimization algorithms to the given problem. The implementation can be done in parallel by many developers as shown in the Figure 1.

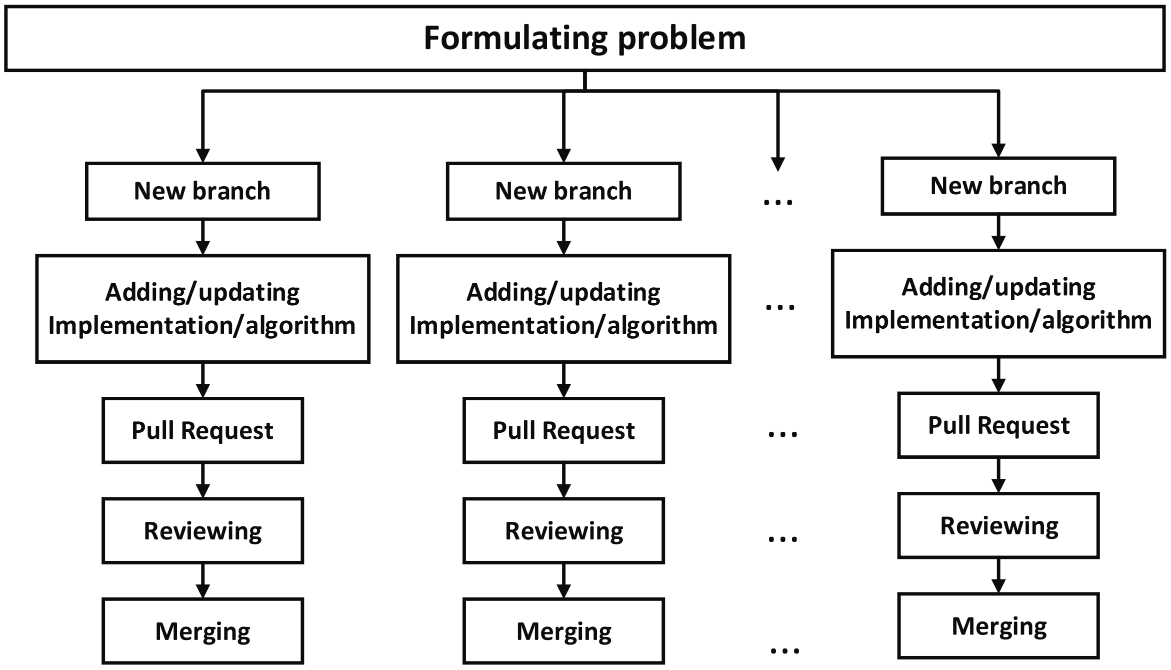


Figure 1. The proposed methodology for extending the library.

Q10: how volunteers, students, developers, researchers can contribute?
The contribution is easy. The project is hosted in the GitHub. Therefore, any contributor can create new branch and add or update the project. Then, follow the GitHub approach where pull requests and issues can be generated. After, the consistency is ensured, the updates will be merged to the main project for next version.

Q11: what are the current version services of this library?
In the current version, I focused on one problem which is partitioning. For this problem, we implemented three algorithms as described in Table 1.

Table 1. The algorithms of the current version 0.0.1

Algorithm	Description
kernighan_lin_0101**(**G**,** initial_partitions**,** attrs**)**	Kernighan-Lin algorithm is well-known bi-partitioning heuristic. This implementation is optimized where many overheads are eliminated. The time complexity of this implementation is O(n^2). This algorithm can have only one parameter which is G (undirected, unweighted, or Edges-weighted Graph) and produce balanced partitions. The initial_partitions parameter is optional.
simulated_annealing_0102**(**G**,** initial_partitions**,** attrs**)**	Simulated annealing algorithm is well-known meta heuristic algorithm. This implementation is developed for graph bi-partitioning problem. This algorithm can have three parameters which are G (undirected, unweighted, or Edges-weighted Graph), and two algorithm-specific attributes (cooling_rate and temperature). The initial_partitions parameter is optional. This algorithm produces balanced partitions.
Hybridization_SA_KL_0103**(**G**,** initial_partitions**,** attrs**)**	This Hybridization technique can help adapting the advantages of the two algorithms (simulated annealing and Kernighan-Lin algorithms). The proposed hybridization can optimize both running time and cost cut quality. This algorithm can have three parameters which are G (undirected, unweighted, or Edges-weighted Graph), and two algorithm-specific attributes (cooling_rate and temperature). The default values for cooling_rate is (0.0001) and for temperature is (10000). The initial_partitions parameter is optional. This algorithm produces balanced partitions.

For any algorithms of partitioning problem, the common input parameters are described in Table 2, the common output data are described in Table 3, and the common functions are described in Table 4.

Table 2. General input parameters for partitioning problem

Parameters	Descriptions
Graph (G)	There are many graph types. The possible input graph for the partitioning problem is listed below: • Undirected and unweighted graph. • Undirected and edges-weighted graph. • Undirected and vertices-weighted graph. • Undirected, edges-weighted, and vertices-weighted graph.
initial_partitions	This parameter is optional. If it is passed, then this parameter will enforce all partitioning algorithms to start partitioning from the initial partitions. If this parameter is not passed, then the initial partition will be generated randomly.
attrs	This parameter is a dictionary data structure. This parameter is designed to carry out algorithm-specific attributes/parameters. It is optional and depend on the algorithm requirements. For example, simulated annealing algorithm required two algorithm-specific parameters. If the dictionary is empty, or not passed, then the algorithm will use the default. Each algorithm should define the default values for each attribute.

Table 3. General output for partitioning problem

Returns	Descriptions
best cost cut	The total weights of cross-partition edges are called cost-cut. The goal is to minimize the number of cross partition edges. This output shows the minimum cost-cut achieved by the current run of the given algorithm.
best partitions	It returns the partitions that have minimum cost cut (best cost cut). It will return iteratable object that contains the nodes labels of each partition.

Table 4. General functions for partitioning problem

Common functions	Descriptions
get_initial_partitions()	This will return iteratable object contains initial partitions.
get_best_partitions()	This will return iteratable object contains best partitions.
get_initial_cost_cut()	This will return a value that shows the cost-cut of the initial partitions.
get_bestt_cost_cut()	This will return a value that shows the cost-cut of the best partitions.

Q12: how this library can be installed?

Use the package manager *pip* to install optimization-algorithms.

```
pip install optimization-algorithms.
```

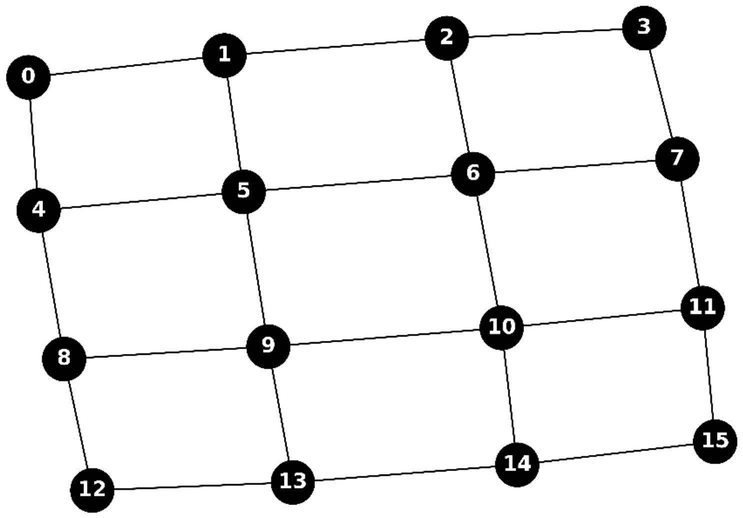
Q13: how this library can be used?

To show how the optimization-algorithms library can be used for partitioning problem, we need a dataset. In this case, we will use two-dimensional grid graph to show how this library can be used for portioning problem. In the grid graph, each node connected to its four nearest neighbors. To generate the grid graph, we implement the following function:

```
import networkx as nx
def generate_grid_2d_graph_dataset(grid_size):
   cols=[(i * grid_size + j, i * grid_size + j+1)  for i in range(grid_size) for j in range(grid_size-1)]
   rows=[(j * grid_size + i, j * grid_size + i + grid_size)  for i in range(grid_size) for j in range(grid_size-1)]
   edgeList=cols+rows
   return  nx.Graph(edgeList)
```

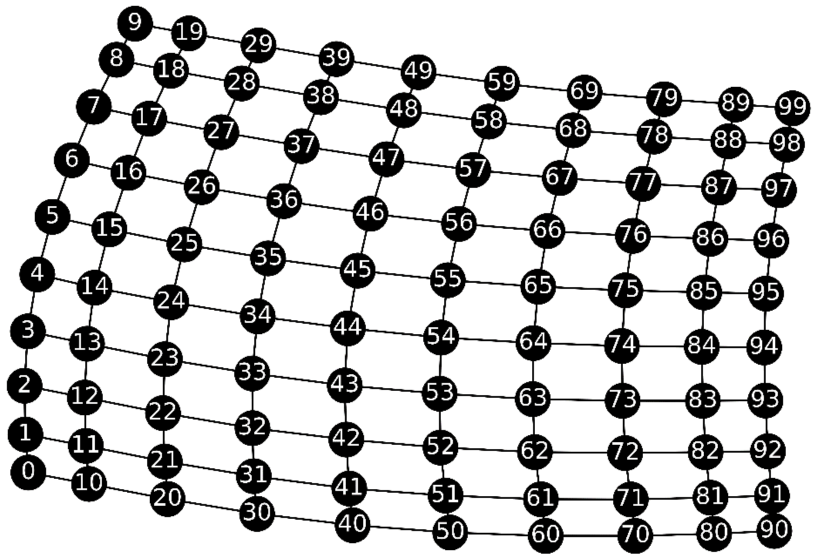
The following code shows how 4X4 grid graph can be generated.

```
G=generate_grid_2d_graph_dataset(4)
nx.draw(G, with_labels=True)
```



The following code shows how 10X10 grid graph can be generated.

```
G=generate_grid_2d_graph_dataset(10)
nx.draw(G, with_labels=True)
```



In the first step, we will import the optimization-algorithms library for partitioning problem where three algorithms for bi-partitioning will be imported. The following code show how these algorithms can be imported.

```
from optimization_algorithms.for_partitioning import kernighan_lin_0101,
                                                     simulated_annealing_0102,
                                                     Hybridization_SA_KL_0103
```

The following code shows how 100X100 grid graph can be partitioning using Kernighan-Lin (KL) with the default parameters.

```
KL=kernighan_lin_0101(G)
```

Output:

```
-------Result Summary---------
initial cost-cut :              9841
Best cost-cut :                 130
Running Time in Second :        76.8
```

The best partitions can be obtained by the following function:

```
partitions=KL.get_best_partitions()
```

The following code shows how 100X100 grid graph can be partitioning using Simulated Annealing algorithm (SA) with the default parameters.

```
SA=simulated_annealing_0102(G)
```

Output:

```
-------Result Summary---------
initial cost-cut :          9837
Best cost-cut :             4454
Running Time in Second :    3.1
```

The best partitions can be obtained by the following function:

```
partitions=SA.get_best_partitions()
```

The following code shows how 100X100 grid graph can be partitioning using Hybridization SA&KL algorithm with the default parameters.

```
SA_KL = hybridization_SA_KL_0103(G)
```

Output:

```
-------Result Summary---------
initial cost-cut :          9902
Best cost-cut :             119
Running Time in Second :    35.5
```

The best partitions can be obtained by the following function:

```
partitions= SA_KL.get_best_partitions()
```

To create initial partitions and pass it as parameter, in this example, we may create the initial partitions as even partition and odd partition where the nodes with even labels will be in one partition, and the nodes with odd labels will be in another partition. The following function can help generate the initial partition.

```
def create_initial_partitions(G):
    even=[i for i in G.nodes if i%2 == 0]
    odd =[i for i in G.nodes if i%2 == 1]
    return [even, odd]
```

The following code is used to prepare the initial_partitons parameters.

```
initial_partitions=create_initial_partitions(G)
```

Now, we can pass this parameter to the three algorithms.

```
KL=kernighan_lin_0101(G, initial_partitions)
```

Output:

```
-------Result Summary---------
initial cost-cut :           9900
Best cost-cut :              111
Running Time in Second :     56.4
```

```
SA=simulated_annealing_0102(G, initial_partitions)
```

Output:

```
-------Result Summary---------
initial cost-cut :           9904
Best cost-cut :              4541
Running Time in Second :     3.0
```

```
SA_KL = hybridization_SA_KL_0103(G, initial_partitions)
```

Output:

```
-------Result Summary---------
initial cost-cut :           9928
Best cost-cut :              121
Running Time in Second :     35.4
```

The following code shows how to play with algorithm-specific parameters such as cooling-rate and temperature parameters of simulated annealing algorithm and hybridization SA-KL algorithm.

```
Attrs={'cooling_rate':0.001, 'temperature':1000}
```

```
SA=simulated_annealing_0102(G, initial_partitions, Attrs)
```

Output:

```
-------Result Summary---------
initial cost-cut :           9845
Best cost-cut :              7686
Running Time in Second :     0.6
```

```
SA_KL = hybridization_SA_KL_0103(G, initial_partitions, Attrs)
```

Output:

```
-------Result Summary---------
initial cost-cut :           9923
Best cost-cut :              103
Running Time in Second :     50.1
```

There are some Tips in using parameters. Simulate annealing and hybridization are different algorithm. It is better to examine each one with different values of the cooling rate and temperature parameters. The hybridization algorithm requires only small values. For Kernighan-Lin algorithm, there are some useful functions that allow user to obtain some useful information. The following shows those functions.

```
KL.get_number_of_iterations()
```

Output:

```
19
```

We can notice that the number of iterations by KL for the given dataset is 19.

```
KL.get_cost_per_iterations()
```

Output:

```
{9900, 3154, 2900, 2836, 2436, 2100, 1900, 1700, 1500, 1300, 1100,
1002, 700, 602, 237, 154, 135, 111, 111}
```