Solution to Planning of resources optimisation problem

Anyou Jiang

June 12, 2021

Summary This short article is to explain a solution, FloorPlanning, to solve the optimization problem of planning of resource. It describes the solution details, the implementation and some simulation results are discussed.

Planning of resources optimisation problem

I have 4 tasks to complete for one unit. I have n units to complete. Each task requires different number of people:

Task 1 - 6 people - 5 days to complete
Task 2 - 3 people - 2 days to complete
Task 3 - 3 people - 6 days to complete
Task 4 - 3 people - 5 days to complete

Constrains are as follows: Task 1 needs to complete before the Task 1 on the next unit. Task 1 needs to be completed before Task 2, Task 3, and Task 4 in the same unit.

So Task 2, 3 and 4 can start at the same time if the people are available.

Input is number of available people (or if it is easier number of teams 3 people = 1 team). For example 6 (2 teams) shoulbe be minimum because 6 are needed for Task 1. After compleating Task 1 they can be split into two teams and one team can start the work on Task 2, 3 or 4 and another the same one Task 2, 3 or 4.

Result should be a list and visual print of the timeline with number of days and tasks

Solution

The above problem is similar to the Floor-Planning problem in EDA design[1]. Each task in each unit can be treated as a hardware module with a fixed rectangle shape. And the constrains between the tasks in the same or in the different units can be treated as the constrains in the positions of the modules during the floor planning.

In the first version of the python implementation, it will using the **skewed slicing tree and its normalized Polish Expression** to represent any possible scheduling result of the tasks. The advantage of the slicing tree structure lies in its simplicity, but it can't represent all the possible scheduling solutions such as those 'non-slicing' solutions. So using slicing tree can provide a faster prototyping and a sub-optimal solution. To get better solution, the other more complex structures such as B^*-**tree** and **Sequence Pair** can be considered in the future versions.

[1] Tung-Chieh Chen. Chapter10 floorplanning. `http://cc.ee.ntu.edu.tw/~ywchang/Courses/PD_Source/EDA_floorplanning.pdf`

The python implementation also follows the **Simulated Annealing(SA) Approach** described in the section 10.2 in the reference[2], but with some simplifications. The SA approach is a meta-heuristic optimization method popular used in *NP-hard* combinational problems such as floor plan and resource scheduling problems where the solution space is huge and non-analytic. It can easily incorporate the constrains and the optimization goal during the search for the sub-optimal solutions. Fig. 1 is an illustration of the SA process. During the iteration in the solution space, many local optima will be encountered, the advantage of the SA is that it can up-hill and climb over the local optimal in order to search for better optimal after hill climbing.

[2] Tung-Chieh Chen. Chapter10 floorplanning. `http://cc.ee.ntu.edu.tw/~ywchang/Courses/PD_Source/EDA_floorplanning.pdf`

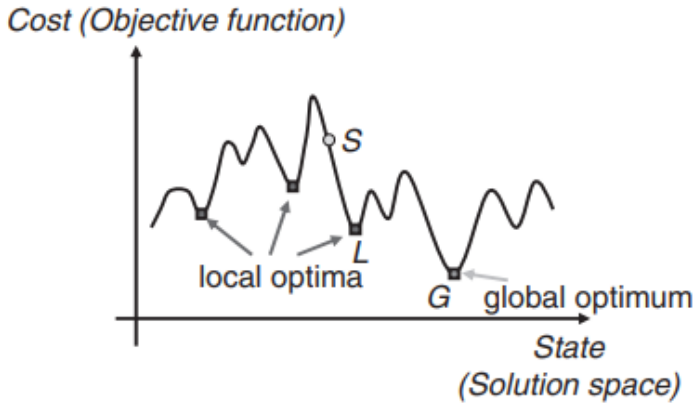


Figure 1: Illustration of the Simulatin Annealing Process

In this resource planning planning problem, the objective during the SA process is to minimize the **aggregated** delay of all the tasks within the constrains in the total persons and total days and the constrains of the precedences between the tasks. Here is an example to calculate the aggregated delay for a simple scenario: there is one unit, with 4 tasks, and Task-1, Task-2, Task-3, Task-4 are scheduled in the sequence order, i.e their starting time is the 0-th, 5-th, 7-th, 13-th day; the completion day is the 5-th, 7-th, 13-th, 18-th day, so the aggregated delay is

$$D = 5 + 7 + 13 + 18 = 43(days)$$

Implementation

The python project to implement the solution is shown in the Fig. 2.

Here is the short introduction to the source codes:

- ResourcePlanning.py : it is the entry file (i.e. main) to run the simulation. One can configure the number of persons and number of days, number of unit for the optimization problem, like the following

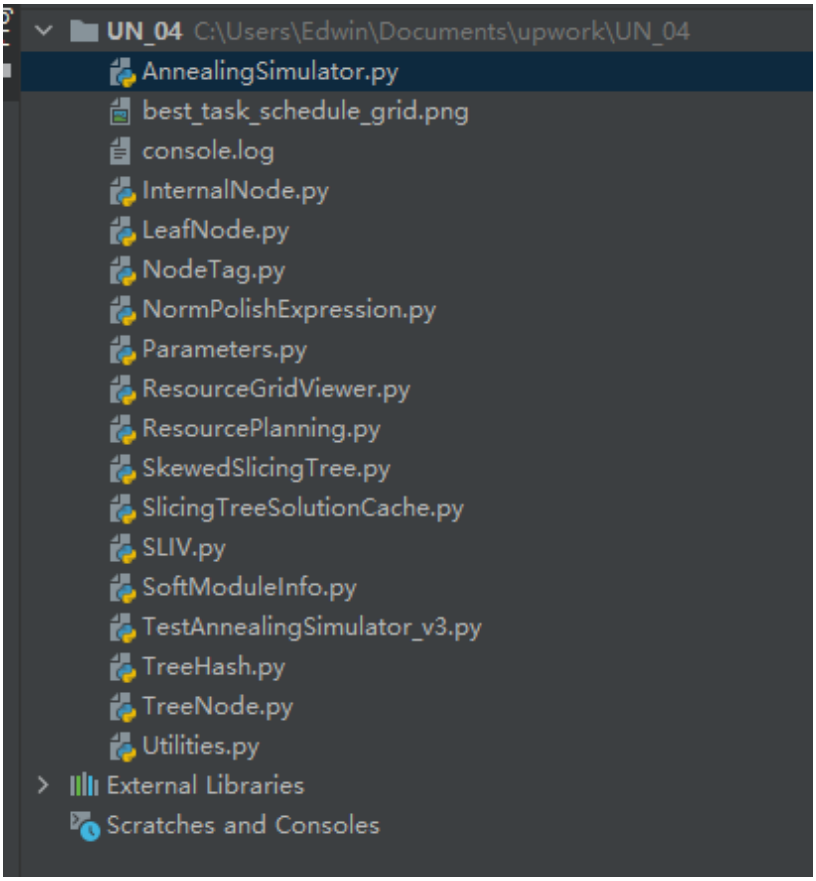


Figure 2: Python project

$ResourcePlanning(maximmum\_persons = 40, maximum\_days = 40, num\_of\_unit = 3)$

- AnnealingSimulator.py: it is the implementation of the SA method, where the key steps are implemented in the function, 'def runSimulation(self)'
- SkewSlicingTree/InternalNode/LeafNode/NodeTag/TreeHash/TreeNode.py: it is the implementation of the skewed slicing tree data structure
- NormalPolishExpression.py: it is the implementation of the normal polish expression for the skew slicing tree
- ResourceGridViewer.py: to plot the final solution into grid plot
- SLIV.py/SoftModuleInfo.py: to support further extensions in support flexible shapes.
- SlicingTreeSolutionCache/Untilities.py: some utilities to improve the run time
- Parameters.py: some basic configuration.

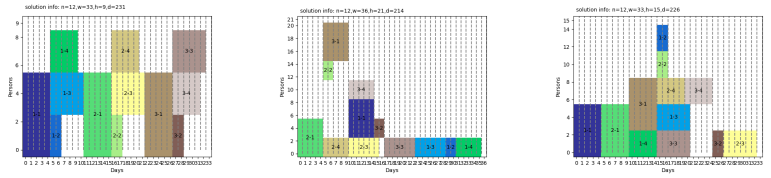
Simulation Result

In the simulation, an experiment with the following parameters in Table [1] are designed.

Parameter	Value
number of unit	3
maximum number of days	40
maximum number of people	40

Table 1: Experiment Parameters

The final solutions from three simulations are shown in the Fig. [3].



(a) The resource scheduling in the 1st simulation

(b) The resource scheduling in the 2nd simulation

(c) The resource scheduling in the 3rd simulation

Figure 3: Suboptimal resource scheduling results from three simulations

As can be seen in the figures, the results from the simulation are suboptimal under the constrains. In left sub-figure in Fig. [3], it has the shortest days to finish the tasks in three units, however, the aggregated delay is 231 day, which is larger than that in middle subfigure which has an aggregation day of 214.

In general, the results from the solutions are not optimal because, in the first version of the implementation, slicing skewed tree is adopted, which has limited the solution search space. To get further optimal results, as stated in the section , more complex structures such as B^*-**tree** and **Sequence Pair** can be considered.

References

Tung-Chieh Chen. Chapter10 floorplanning. `http://cc.ee.ntu.edu.tw/~ywchang/Courses/PD_Source/EDA_floorplanning.pdf`.