Local Search Algorithm

Barak-Nadav Diker

October 24, 2024

Contents

1 Website

For Reading the docs with beautiful web please visit website better graphics

2 Quickstart

In order to run the program one should clone the repo and run the following command

```
python local_search_algorithm.py --input_path test/input/input.py
↪  --output_dir test_dir
```

the `input_path` should point to a python file for example input.py and should have the following content

```
m = 6
k = 5
processing_arr = [2, 3, 3, 4, 4, 4, 5, 5, 5, 5]
```

if one get stuck he can read the docs via the command

```
python  local_search_algorithm.py --help
```

One should note that the final solution to the problem is in the `output_dir` under the name `output_score.txt` ,
also there is more output contents

3 Visualizations

for animation creation one can use the following command

```
python local_search_algorithm.py --input_path
↪  possible_solutions/sol3/input/input.py --output_dir
↪  possible_solutions/sol3 --visualize_simple --animation_flag
```

This will create a new folder in `output` directory called `animation` with GIF

4 Abstract

The Project try to solve the following schedule problem
Scheduling with cardinality constraints.

4.1 Input

An integer number of (identical) machines m 2. A set of n jobs J = {1, 2, . . . , n}, where job j has an integer processing time $p_j > 0$. A positive integer parameter k.

4.2 Goal

Find a partition (assignment) of the jobs into subsets, I_1, I_2, . . . , I_m, where for any 3 i m, it holds that $|I_i|$ k (a cardinality constraint of k is enforced for all machines except for possibly the first two machines, that is, every machine whose index is at least 3 can receive at most k jobs).

5 Command to test

```
sudo python launcher.py --input_path
↪  possible_solutions/lea_inputs/question_3/2l/input/input.py
↪  --output_dir possible_solutions/lea_inputs/question_3/2l
↪  --visualize_simple --animation_flag
```

6 Algorithm Local Search Barak's heuristic

How the Final algorithm works Given an array of jobs which I'll call $\text{processing}_{\text{arr}}$, number of machines which will be marked with m , and a maximus job number for machine of greater index of 3 which be be marked as k

The following process will take place

- Create array of array of size m which named machines
- Put all the jobs on machine 0 (The first machine)
- Create PseudoArray of machine jobs of size m which named previous-machines
- While machines not equals to previous-machines
  - While a move occured
    - from the machine which has the highest **processing time** move the longest possible job to the machine which has the lowest processing time under the problem constraint
  - While a move occured
    - from the machine which has the highest **job count** move the longest possible job to the machine which has the lowest **job count** under the problem constraint
  - While a move occured
    - from the machine which has the highest **processing time** move the longest possible job to the machine which has the lowest processing time under the problem constraint
  - For all 2 machines and all job in the first machine and all job in second machine exchange the 2 jobs and if the overall processing time has decreased apply the change and exit the loops

7 Improvements and Significal developmets

- At first I have tried to move jobs under constraint until there is not more moves and then exchange once,
  - This Procedure was not effective
- I've tried couple of different combinations of movement and exchanges that were not so great