# Local Search for scheduling

Ebraheem Kashkoush

May 17, 2020

## 0.1 Abstract

In this project we use Local Search to solve scheduling problime, the problem is :
Scheduling on restricted uniformly related machines.
Input:An integer number of machines m $\geq$ 2. A set of n jobs J={1,2,...,n} where job j has an integer processing time $P_j > 0$ Machines speeds $s_i \in \{1, 2, 4\}$ fori= 1,2,...,m
Goal:Find a partition (assignment) of the jobs into non-empty subsets $I_1, I_2, ..., I_m$
Objective:...
.

Note: it's not the best way to solve the problem using Local Search but i use this problem to build example of using Local Search we can get better soloution by using compination of B&B and Local Search for example.
in my soloution some time it enter to local minimum for that reason we will discuss on the future work how we can solve this problem and also how to optimize the algorthim run time.

## 0.2 algorthim

### 0.2.1 initial soloution

- first we read the input files,we have two inputs the first one for tasks each line in input file has task time value ,and the second file for machines each line have the speed of new machine.
  we enter the input into two vectors 1- vector<Node> J , and 2- vector<machin> M ,where J hold tasks (Node hase two value : index and task time) ,and M hold machines
  each machine has :
  int speed :- hold machine speed {1,2,4}
  int TasksTime; :- hold the total tasks time
  int index :- hold machine index
  std::map<int,Node> Tasks :- map that hold the Tasks chossed for the machine by scheduler.
  **Note:** We chosed hash map becouse we will try to change tasks bettwen each two posible machines ,and hash map has best complixity for finding task by index, becouse must of the time we only check and did not swap the tasks bettwen one machine to othere (only if swaping the tasks from one machine to other make us more close to the goal

)we change the tasks bettwen two machines (and this probability is unlikly).
but as result we have a problem when we want to chose k tasks frome $machine_1$ and r tasks from $machine_2$ we dont now witch indxes in each machine so we build an 2D array hold the indxes ,its ok to hold array becouse only if we swap the tasks we remalloc the rows of this two machine and update the row and we said before that probability is unlikly

- then we enter the tasks into Min_Heap .
  and while te MinHeap is not empty we do sequentially loop for machines each time we gave each machine tasks count the same as the machine speed for if the machine speed 1 we give at 1 task and if the machine speed 2 we give it 2 tasks etc... .
  that was the initial solotion and on that soloution we applay the LocalSearch algorthim

### 0.2.2 Local Search of two machines

givin two machines m1 and m2 , we want to find the best tasks that if we swap them bettwen m1 and m2 we get the best Improvement to achive the Goal .
where the goal is :
let assume that we chose $k$ tasks from $m1$ and $r$ tasks from $m2$ , and A,B is the time of machines $m1, m2$ after swaping the tasks ($r$from$m2$ ,$k$ from $m1$)and X,Y is the time before sewaping the tasks .
we chose $k$ task from all posible compination to chose $k$ tasks from $m1$ and $r$ tasks from all posible compination in $m2$ such that give as: .
max{A,B}<max{X,Y} or ( max{A,B}=max{X,Y} and (A+B)<(X+Y)) .
that mean we go throw the compination sequentially if the new $r, k$ give us maximum machine timing less than the best soloution for $m1, m2$ or the same maximum but the sum of A+B<X+Y we chose the new $r, k$ as the best soloution for now and save $r, k$ and check the other $k, r$ compinition with the new best solution.
Note: the ( max{A,B}=max{X,Y} and (A+B)<(X+Y)) equation help us to give high priority to machines with high speed.
Above we describe one iteration but we repeat the iteration untill we can not find $r, k$ that give us best soloution if we swap them. and then we stop the Local search bettwen $m1$ and $m2$

### 0.2.3 All posible combination of r,k

as we said above we have a 2D array ,where each row $i$ we hold the tasks in $mashine_i$ ,that mean we have $m$ rows wehre $m$ is machines count ,and each cell on $row_i$ hold $machine_i$ task index.
we start first solve the problem of choosing $k$ tasks from machine. given array we want to get all the different combination to reorder the array as we know in math its $\binom{n}{k}$ problem where $n$ is the tasks count and $k$ is the number of tasks that we want swap with the other machine , that mean we wan to chose $k$ tasks from $n$ tasks.
given array hold machine index tasks that problem can done verey easly using recursion
if we creat array with size $k$ we want to relaod to it one combination each time and once the array full we have new combinition , and we have index that point to the next empty cell we can done the problem with this two recursive recall

// current is included, put next at next location
data[index] = arr[i];
combination(arr, n, k, index + 1, data, i + 1);

// current is excluded, replace it with next (Note that
// i+1 is passed, but index is not changed)
combinationUtil(arr, n, k, index, data, i+1);
on other word we have two option to satisfy and set arr[i] in data[index] or to try to give him another i' so we find $j = i + c, c > 0$ to set data[index]=arr[j];

### 0.2.4 combination of two machines

i go throw all posible cobinations of two machines twice on deferent order the code is so easy so writing the code more sample than to dscribe it :

```
for (int offset = 1; offset < M.size(); offset++) {
    for (int i = 0; i < M.size() / 2; i++) {

      GetBestOfNxM(i * 2, n, (i * 2 + offset) % M.size(), m,
   0, d, 0, true);
      if(!GetBestOfNxMbool)
      SwapmTasks(NxMcom1Best, i * 2, NxMcom2Best, (i * 2 +
   offset) % M.size());
      flag = flag && GetBestOfNxMbool;
```

```
8
9
10      }
11     }
12
13     for (int offset = 0; offset < M.size(); offset++) {
14      for (int i = 0; i < M.size() ; i++) {
15
16       GetBestOfNxM(offset, n, i, m, 0, d, 0, true);
17       if (!GetBestOfNxMbool)
18        SwapmTasks(NxMcom1Best, offset, NxMcom2Best, i );
19       flag = flag && GetBestOfNxMbool;
20      }
21     }
```

### 0.2.5 all togther

above we see how we chose pair of two machines and how we applay LocalSearch on this two machines ,weher we see how we chose k tasks from n and we check each combilition of $m1(firstmachinechosek)$ with all the cobination of $m2(machine2chosingrtasks)$.
so we can write one function $LocalSearchNxM(r,k)$ and applied LocalSearch that swap $r$ tasks from first machine with $k$ from machine 2 and we go throw all compinition of two machine like we said above
and this is the main LocalSearch Function:
Where:
LevelZero(): is function implement local search that only pass one task each time from machine1 to machine2 with the same rolles above.
LocalSearchNxM(r,k): function implement localsearch where we want swap each time $r$ tasks from $m1$ with $k$ tasks from $m2$

```
1  void LocalSearch() {
2   bool flag = true;
3   bool temp;
4   while (flag) {
5
6    flag = flag && LevelZero();
7    for (int i = 1; i <= maxLevelSearch; i++) {
8     for (int j = i; j <= maxLevelSearch; j++) {
9      if (j == 1 && i == 1) {
10      temp = LocalSearchNxM(i, j);
11      if (temp) {
12       i = 4; j = 4;
13      }
14      flag = flag && temp;
15     }
```

```
16      else {
17
18       if (j == 1) {
19        if(TasksTable[i-1][maxLevelSearch]>0)
20         temp = LocalSearchNxM(i, j);
21        if (temp) {
22         i = 4; j = 4;
23        }
24        flag = flag && temp;
25       }
26       else {
27        if (TasksTable[i][j-1] > 0)
28         temp = LocalSearchNxM(i, j);
29        if (temp) {
30         i = 4; j = 4;
31        }
32        flag = flag && temp;
33       }
34      }
35
36
37
38
39
40     }
41    }
42
43
44
45     flag = !flag;
46
47   }
48  }
```