Scheduling on restricted uniformly related machines.

Ebrahim Kashkoush

1 Abstract

in this project we try to solve scheduling problem using three Heuristics Local search,Branch and bound and genetic heuristic,we write three unattached projects in each one we use one heuristic,we can get better and faster algorthim by Combine more than one heuristic or by using more complex algorthims like evaluate pairs of chromosomes instead of chromosomes ,and use more complex target function in genitic. but the main idea of the project was to learn the basic concept of the following heuristics.

The problem is Scheduling on restricted uniformly related machines. where the Input is An integer number of machines m ≥ 2. A set of n jobs J=$\{$1,2,...,n$\}$ where job j has an integer processing time $P_j > 0$ Machines speeds $s_i \in \{1, 2, 4\}$ for i= 1,2,...,m , we solve the same problem in the three heuristics

2 Local Search

The local search algorthim It consists of several steps we will Explain the algorithm step by step

2.1 pre preparing(initial solution)

to start running the local search algorthim first we need initial solution. we have as input two file the first file is for tasks where we have on each line task time (integer value) , the second file is for machines where each line contains machine speed (integer 1,2,4). from the input we build two vectors the first one hold the tasks and the second hold the machines . after having data structure hold the tasks and the machines we should Hand out the tasks into the machines and make initial soulation , so we need greedy algorithm its not important how many the greedy algorthim is good. also if we hand out the task randomly at the end we will have the same result , so for convenience we insert the numbers to minmum heap to sort the tasks and then we Turned on the tasks into machines In a fair way (In a circular) and on each time we give the same amount tasks as the current machine speed value to the machine , for example if we have machine with speed 2 we will give it two task each time ,and if the machine speed 4 we will give it 4 tasks each time . we do that while we still have tasks, and when we hand out all the tasks we have an initial solution.

2.2 Local Search for two machines

on this subsection and on the next subsections we will solve small problems and then we will combine all of them together in the last subsection creating the full problem solution.

the current problem is given two machines m_1 and m_2 we want to find the best tasks from m_1 and m_2 that if we swap them bettwen m_1 and m_2 we get the best Improvement to achive the Goal . so now we should define what is the goal that we want to achive.
givin two machines m_1 and m_2 where the time of m_1 is X and the time of m_2 is Y . let assume that we chose k tasks from m_1 and r tasks from m_2 now we should evaluate the time after swapping the tasks (simulation) between the two machines by passing k tasks from m_1 to m_2 and removing the k tasks from m_1 ,and the same by passing r tasks from m_2 to m_1 and removing the tasks from m_2. lets define the time of m_1 after the change is A and the time of m_2 after the change (swaping the tasks) is B . the our goal that we want to achive:
(max{A,B}<max{X,Y} || (max{A,B}=max{X,Y} and (A+B)<(X+Y)))
that measn if we save the best soulation and we find new soultion by swaping k, r tasks bettwen m_1 and m_2 and that soulation better than the best soulation for now, we save it as the best soulation and we save r, k tasks and then we check the other k, r compinition with the new best solution after we check all possible to choice r, k we have the best r, k that by swaping them we achive our goal NOTE(k and r is constant).
the idea of adding (max{A,B}=max{X,Y} and (A+B)<(X+Y)) is to give priority to machines with high speed we want to fill machines with high speed also if the final soulation still the same (max{A,B}), because that help us to find bigger task to swap in the future.

2.3 All posible combination of r,k

as we saw in previous section we need to go through all possible combinations of choosing k tasks from m_1 (the same about r) as we know from mathematics we have $\binom{n}{k}$ different combination. where n is the number of tasks on m_1 and k is the number of taskt that we want to choose from m_1 to swap them.
we can solve the combination problem easily by using recursion and array with size k (int data[k]) , we go through all the n tasks we start from first task (first function call) and we call the function recursively twice the first call with array data that we add to it task1 and the second one with array data without adding to it task1, in general in the p funcion *call* we call the function recursively twice ,the first time with adding $task_k$ to data array and the second with out adding it. and when data array full we fill al the k cells we have new combination.
in other words the idea that each task has two possibilities the first one is to be part of the combination and the second its not part of the combination.

2.4 combination of two machines

i go throw all possible 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++) {

      // the index of first machine i * 2
    // the index of second machine (i * 2 + offset) % M.size()
    //M.size() is number of machines
     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;


    }
   }

   for (int offset = 0; offset < M.size(); offset++) {
    for (int i = 0; i < M.size() ; i++) {

     // the index of first machine offset
    // the index of second machine  i
    //M.size() is number of machines
     GetBestOfNxM(offset, n, i, m, 0, d, 0, true);
     if (!GetBestOfNxMbool)
      SwapmTasks(NxMcom1Best, offset, NxMcom2Best, i );
     flag = flag && GetBestOfNxMbool;
    }
   }
```

2.5 all togther

we show above how to go throw all posible combination of pair of two machine. and on each combination of two machines how to go throw all possible combination of k tasks from m_1 and r tasks from m_2 (where k and r is constant) and how to choose the best combination of choosing k,r(how we applay LocalSearch on this two machines) (we check all the combination of choose k from $machine_1$ with all the combination of choose r from $machine_2$ "each combination from $machine_1$ with all the combination of $machine_2$ ")

so we can write one function $LocalSearchNxM(k, r)$ and applied LocalSearch that swap k tasks from first machine with r from second machine and we go throw all compinition of two machine like we said above
this is the main LocalSearch Function:
Where:
LevelZero(): is function implement local search that only pass one task each time from $machine_1$ to $machine_2$ with the same rolles above.
LocalSearchNxM(k,r): function implement localsearch where we want swap each time k tasks from m_1 with r tasks from m_2
if the there is no improvement in the current level we did not run the next level , and if there is no level on this itertion has improvement we stop the while .and after we stop we print the result

```
void LocalSearch() {
 bool flag = true;
 bool temp;
 while (flag) {

  flag = flag && LevelZero();
  for (int i = 1; i <= maxLevelSearch; i++) {
   for (int j = i; j <= maxLevelSearch; j++) {
    if (j == 1 && i == 1) {
     temp = LocalSearchNxM(i, j);
     if (temp) {
      i = maxLevelSearch+1; j = maxLevelSearch+1;
     }
     flag = flag && temp;
    }
    else {

     if (j == 1) {
      if(TasksTable[i-1][maxLevelSearch]>0)
       temp = LocalSearchNxM(i, j);
      if (temp) {
       i = maxLevelSearch+1; j = maxLevelSearch+1;
      }
      flag = flag && temp;
     }
     else {
      if (TasksTable[i][j-1] > 0)
       temp = LocalSearchNxM(i, j);
      if (temp) {
       i = maxLevelSearch+1; j = maxLevelSearch+1;
      }
      flag = flag && temp;
     }
    }






   }
  }




   flag = !flag;

 }
}
```

3 Branch and bound

on this heuristic we should build tree, each node on the tree have lower bound and higher bound , each edge bettwen father and son represent handing out task to the son, where each node represent machine , so each father has at most children's as the number of machines, so the path from the root to each node in the level-k (node with deapth k) represent handing out the first k task where $edge_j$ represent handing out $task_j$, but we will not build all the tree because if there node that have lower bound equal or greater than any upper bound on any node on the tree there is no need to continue to his son's becouse we will not get better soultion than the node with the upper bound, for that reason we have also a global upper bound that his value is the minimum value on all the upper bounds that we have on all the nodes that we build to that point.(on building each node we update his value if the node upper bound is less than the global). upper bound can be greedy algorithm result but unlike Local search the greedy algorithm is important because bad greedy algorithm mean that we need to build more nodes on the tree, and as result we will have to wait

more time to get the solution, so here for example its good idea to set the result of local search as upper bound but we did not do that because most of the time we get optimal soultion using local search and our goal to learn the heuristic and to find the best way to calculate the upper and lower bounds.
so our greedy algorthim is LPT (Longest Processing Time Algorithm) . also the sequence of handing out the tasks on branch and bound algorthim is important by handing out the big tasks first can help us to build less nodes because the lower bound grow and get bigger faster, also this issue in line with LPT ,so $task_1$ is the biggest task and $task_n$ is the smallest task , and on each path $edge_p$ represent handing out the $task_p$.
At the end we want to run the program on personal computer and our challenge to build algorthim with good Space complexity (use less amount of memory space) . if we use BFS tree traversals algorthim we need a huge memory because on the first levels of the tree we can't determine if the path is good and we need to build and save more nodes that we can't remove node before we make sure that this path is bad path and we will not continue to his son's, also if we be satisfied to save only one level (depth) each time (save the last level of depth nodes and one each nood save the path from the root (each task to each machine)) the amount of nodes could be huge , despite that the BFS does not guarantee better soultion, on other hand using DFS we can build solution with linear space complexity and also that help the global upper bound to decrease faster because we cant to determine tight upper bound if we still have many tasks to handing out because the upper bound will be for all possible different solution of handing out the other tasks (maximum possible upper bound), so going deeper help us make tighted upper bound, and DFS go deeper fast. we can build solution using DFS that his maximum space complixty $m * n$ where m is the number of machines and n is the number of tasks.
we have customized DFS in our soluation the defult is to going to the worst son first (son's with the highest time first) we have the option going to the best son's first (have lowest time first (where the time is the time of the worst machine)) by deleting (#define BadFirst) on file scheduling.h . the idea of choosing best son's first is to get tight upper bound faster and that help us to build less nodes, on the other hand the idea of choosing worst son's first is to make the lower bound grow faster and as result we build less nodes, we try the three options without sorting(orginal DFS) ,bad first and best first we get the result of bad son's first is the best. and we also have to option to sort the tasks at the beginning like we said above big tasks first or run without sorting by removint the define (#define _WithSort) but as we explain with sort is more better .