Summary

The Job Scheduling problem is an important area of research. It is very popular among many researchers from all over the world. However, the Job Scheduling problem is NP-complete; it is impossible to find an algorithm which solves the aforementioned problem in a deterministic way and in polynomial time. In that paradigm the main task of research is to find algorithms which give as good as possible results in a reasonable time-frame. Beyond a doubt, it is a great opportunity to check the effectiveness of Job Scheduling algorithms and to compare them with solutions of other researchers. These are meaningful competitions, such as the recent Job Scheduling Competition on the TunedIT platform. In this paper we are going to present our novelty algorithm based on multi-criteria optimization, which achieved second place during the aforementioned competition. The key point of our approach is the usage of some additional heuristics which allowed us to arrive at such good results.

Keywords: Job scheduling, Multi-criteria optimization

1. Introduction

Scheduling problems arise in a variety of settings [1–2]. Generally, scheduling problems involve jobs that must be scheduled on machines subject to certain constraints to optimize some objective function. The goal is to specify a schedule that determines when and on which machine each job is to be executed. Scheduling theory is concerned with the formulation and study of various scheduling models and the development of associated solution techniques [3–4]. Some widely studied classical models are single machine, parallel machine, flow scheduling and job scheduling models [5–7]. The job scheduling problem belongs to the set of problems classified as NP Hard Problems [3, 8–10]. The search-based approach to the job scheduling problem is based on the exploration of a feasible solution space to identify an optimal solution. Adaptive search algorithms like Genetic Algorithms [11–12], Tabu Search [13–14] and Simulated Annealing [15–16] have been applied to this domain with success and in many cases are capable of providing optimal or near optimal solutions. Many researchers defined the job scheduling problem as multi-objective optimization task [17–18].

In our approach we utilized multi-criteria optimization [19–20] and some heuristics to deal with problem of job scheduling.
1.1. Problem definition

This problem is very well known. As far as I know, it was first presented by Graham in 1966 [22].

Job scheduling is an optimization problem in computer science and operations research in which jobs are assigned to resources at particular times.

The most basic version is as follows. We are given \( n \) jobs \( J_1, J_2, \ldots, J_n \) of varying sizes, which need to be scheduled on \( m \) identical machines, while trying to minimize the makespan. The makespan is the total length of the schedule (that is, when all the jobs have finished processing).

![Figure 1. General job scheduling task](image)

There are many variations of the problem. Below we described the task of the challenge “Algorithm for Job Scheduling and Task Allocation under Constraints” [21].
The task consists of assigning workers to particular tasks with the minimization of cost of all performed tasks and delay reduction: in other words, maximization of profit and minimization of lateness. All tasks (Tasks) require some skills (Skills) and have a specified complexity (Quantity) and specified execution time (Deadline). Each worker has specified skills (Worker_skills), which allow him/her to perform a specified quantity of work during the work day (Daily_quantity). In addition, it is important to take into account the cost of work for all workers and make the assumption that on the same task more than one worker can work at the same time. The general diagram of Job Scheduling is presented in Fig. 1.

2. Algorithm Description

Our algorithm is based on some heuristics which allow us to plan the jobs in the optimal way. At the beginning of the algorithm execution, all tasks were grouped into subgroups. Tasks are grouped based on skills and sorted by means of execution time. Each group also contains a list of users with appropriate skills. The brief description of our algorithm which reached second place in [21] consists of the following steps.

In the first step, delayed tasks were sorted in the increasing order of quantity. The given tasks were completed by all users at the same time in order to minimize the average lateness. The users with the biggest coefficient \(\text{quantDaily} / \text{costDaily}\) were the cheapest. Hence, they were assigned to the task which demanded the shortest execution time. In case of the impossibility to finish the task on time the next user was assigned to the same task: the task that requires any amount of time smaller than 1 is usually selected to be completed. It is possible that from time to time we have to consider tasks which are scheduled (planned) to do later. For instance, we can make the assumption that we have 10 workers with the same skill, and for the first 5 days only 3 workers have to work in order to finish the work on time. It is possible that on the 6th day we have so much work that not even all workers working together will be able to finish it. One of the possible solutions to that problem is to do part of the work earlier. We can then leave a proper quantity of work for the 6th day. The best way to avoid such situations is to calculate the daily quantity of work for each skill at the beginning of each day.

2.1. Implementation details

In Fig. 2 we can see details of the job scheduling classes. The Job class is the class which describes part of the Task done by the Worker. The Worker class consists of the following parameters:

- \(\text{timeLeft} = 1\) – (time of work in the current day); this value is set at the beginning of each day \((\text{timeLeft} = 1)\) and decreases during algorithm execution,
- \(\text{currentDay}\) – current day of work for the considered Worker.

The next class is the Skill class, which consists of the following parameters:

- \(\text{Tasks}\) – sorted list of tasks to do (unfinished tasks), that require considerable skill,
- \(\text{maxDailyQuantity}\) – maximum quantity of all tasks (related with considered skill) that can be done during the current day; this variable is calculated at the beginning of each iteration (at
the beginning of each day) and decreases during the algorithm’s execution; this value cannot be larger than the sum of the quantity that all workers (related with considered skill) can do during one day,

- **Workers** – all users related with considered skill (workers who have appropriated skills to do the tasks from the Tasks list); sorted list – the first one is the user whose work is the cheapest (has the bigger coefficient: \( \text{DailyQuantity/DailyCost} \)).

The JobScheduling class has the following parameters:

- **MAX_DEADLINE** – maximum deadline of all tasks,
- **coefficient** – this coefficient is utilized to choose tasks in the current iteration; the value of this coefficient is set at the beginning of each day (coefficient = 0) and increases during the algorithm’s execution.

![Figure 2. Job Scheduling](image)

In the next subsection we describe the schema of our algorithm step by step.
2.2. Schema of Algorithm

The schema presented in the paper shows only the main idea of the algorithm; clearly, many important details are included in the source code [21].

In Fig. 3, we can see the following points of the algorithm procedure:

1. **Load Configuration**
   - During configuration loading a list of tasks, workers and skills is created. Each task is added to an appropriate skill object. Each user is added to an appropriate skill object (if the user has 3 WorkerSkills, he/she is added to 3 skill objects) – the user is inserted into the Workers’ list in the appropriate place.

2. **Set** \( \text{timeLeft} \) **for each worker,**

3. **For each** Skill **set** \( \text{maxDailyQuantity} \),

   - \( \text{skillsStatistics}[s][d] \) – sum of quantity of all tasks for skill \( s \) on day \( d \) (if task \( t \) should be finished on day \( d \), the quantity of the task is added to: \( \text{skillsStatistics}[s][d] \), \( \text{skillsStatistics}[s][d+1] \),..., \( \text{skillsStatistics}[s][\text{MAX\_DEADLINE}] \));
   - \( \text{MAX\_DEADLINE} \) – maximum value of deadlines of all tasks;

   if (\( \text{day} < \text{MAX\_DEADLINE} \)) then
   for (int \( d = \text{day}; d < \text{maxDeadLine}; d++ \)) do
     if (\( \text{skillsStatistics}[s][d] / (d - \text{day} + 1) > \text{maxDailyQuantity} \)) then
     \( \text{maxDailyQuantity} = \text{skillsStatistics}[s][d] / (d - \text{day} + 1) \);
     end if
   end for
   else
   \( \text{maxDailyQuantity} = \text{skillsStatistics}[s][\text{MAX\_DEADLINE} - 1] \);
   end if

   - where
     \( \text{day} \) – current day;
     \( \text{maxDeadLine} \) – maximal value of deadline of all tasks of currentSkill,

4. **Assign 0 to “coefficient”,**

5. **Sort skills,**

6. **Take the first skill and assign to currentSkill,**

7. **Assign first task to currentTask,**

8. **Remove currentTask from currentSkill.Tasks,**

9. **Assign workers to do the currentTask.** At this moment we make a decision regarding which workers can be utilized to do the currentTask. If (currentTask.Deadline – day) <= 1, all workers from the Workers list can be utilized; otherwise only “the cheapest” users are considered. Workers who have been chosen are sorted due to specific criteria (optimization task for the sort procedure is defined in the next subsection).

10. **Insert task to currentSkill.Tasks in the appropriate place,**
11. Increment coefficient,
12. \( \text{day}^{++} \).
   Steps of the algorithm with multi-criteria optimization were bolded. Optimization tasks for these steps we will define in the next subsection.
2.3. Multi-Criteria Optimization in Our Algorithm

In Steps 5, 9 and 10 of our algorithm we have utilized multi-criteria optimization. We have defined three optimization tasks for sort procedures. Each of them has a defined different preference model (preference relation). In our opinion, the approach which utilizes only one aggregated criterion is less effective and flexible than multi-criteria optimization.

**Optimization task for sort skill procedure** (Step 5 of the algorithm):

\[\text{statistics}[s] – \text{sum of quantity of all tasks (for skill } s)\text{, which are delayed};\]
\[y, z – \text{skill objects};\]
\[y1 – \text{minTimeLeft for skill } y;\]
\[y2 – \text{statistics}[y];\]
\[z1 – \text{minTimeLeft for skill } z;\]
\[z2 – \text{statistics}[z];\]
\[R = \{(y, z) \in X \times X; y1 < z1 \lor (y1 = z1 \land y2 < z2)\}\]

If relation \( R \) contains \((y, z)\) it means that \( y \) is “better” than \( z \).

**Optimization task for workers sort procedure** (Step 9 of the algorithm):

\[\text{task} – \text{considered task } (\text{currentTask});\]
\[\text{quantity} – \text{task quantity to do } (\text{currentTask.Quantity});\]
\[\text{day} – \text{current day};\]
\[\text{taskEnd} – \text{current time of end work in } \text{day} \text{ for } \text{task};\]
\[y, z – \text{workers};\]
\[y1 – \max (\text{quantity}/y2 + y3, \text{taskEnd});\]
\[y2 – \text{dailyQuantity} (\text{for currentSkill}) \text{ for worker } y;\]
\[y3 – \text{current time of end work in } \text{day} \text{ for worker } y (1 - y.\text{TimeLeft});\]
\[z1 – \max (\text{quantity}/z2 + z3, \text{taskEnd});\]
\[z2 – \text{dailyQuantity} (\text{for currentSkill}) \text{ for worker } z;\]
\[z3 – \text{current time of end work in } \text{day} \text{ for worker } z (1 - z.\text{TimeLeft});\]
\[R = \{(y, z) \in X \times X; y1 < z1 \lor (y1 = z1 \land y2 < z2)\}\]

**Optimization task for the task sort procedure** (Step 10 of the algorithm):

\[y, z – \text{tasks};\]
\[\text{day} – \text{current day index};\]
\[y1 – \text{deadline of task } y;\]
\[y2 – \text{current time of end work of task } y \text{ in } \text{day};\]
\[y3 = y1 - \text{day} - y2;\]
\[y4 – \text{quantity of task } y \text{ (remaining quantity to do)};\]
\[z1 – \text{deadline of task } z;\]
\[z2 – \text{current time of end work of task } z \text{ in } \text{day};\]
\[z3 = z1 - \text{day} - z2;\]
$z_4$ – quantity of task $z$ (remaining quantity to do);

$$R = \{(y, z) \in X \times X; (y_3 \leq 0 \land z_3 > 0) \lor (y_3 \leq 0 \land z_3 \leq 0 \land y_4 \leq z_4)$$
$$\lor (y_3 > 0 \land z_3 > 0 \land y_1 \leq z_1) \lor (y_3 > 0 \land z_3 > 0 \land y_1 = z_1 \land y_4 \leq z_4)\}\}$$

If relation $R$ contains $(y, z)$, it means that $y$ is “better” than $z$ (worker $y$ will appear before worker $z$ on the list).

The results for our algorithms with real data are reported in the next section.

3. The Results of Our Experiments

We have carried out a series of experiments on real data – see Tab. 1- from the TunedIt [21] platform. In Tab. 1 we have the $\text{AverageLateness}$ which is calculated over all tasks. In case the task is finished on time or is finished earlier, the lateness is equal to zero. The Profit is calculated in a simple way, as a difference between the total value of tasks and the cost of doing them according to the generated plan. The Score has been chosen in such a way that the $\text{AverageLateness}$ occupies 3 higher digits after the comma, while $1000\text{profit}$ typically occupies 4–6 digits after the comma.

We won second place on the leader board – see Tab. 2- our final result was the same as the winning result and equal to 0.043878. The detailed results are presented in Tab. 1.

$$Score = \text{RoundedAverageLateness} + \text{ScoreFromProfit}$$

$$\text{RoundedAverageLateness} = \frac{\text{AverageLateness} \times 1000}{\text{double}1000}$$

if (Profit <= 1000000) then
    ScoreFromProfit = 0.0000999;
else
    ScoreFromProfit = (1/Profit) * 1000;
end if
Table of examined data sets – all from TunedIT Job Scheduling competition [21]

<table>
<thead>
<tr>
<th>Data</th>
<th>No. of Users</th>
<th>No. of Skills</th>
<th>No. of Tasks</th>
<th>Average lateness</th>
<th>Profit Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.txt</td>
<td>7405</td>
<td>1045</td>
<td>463175</td>
<td>0.045381</td>
<td>5052319</td>
</tr>
<tr>
<td>2.txt</td>
<td>3660</td>
<td>1040</td>
<td>453690</td>
<td>0.045200</td>
<td>4931228</td>
</tr>
<tr>
<td>3.txt</td>
<td>5932</td>
<td>438</td>
<td>419992</td>
<td>0.043596</td>
<td>4577492</td>
</tr>
<tr>
<td>4.txt</td>
<td>8290</td>
<td>1467</td>
<td>483596</td>
<td>0.045417</td>
<td>5267677</td>
</tr>
<tr>
<td>5.txt</td>
<td>3932</td>
<td>532</td>
<td>423808</td>
<td>0.044028</td>
<td>4618708</td>
</tr>
<tr>
<td>6.txt</td>
<td>2019</td>
<td>1168</td>
<td>407734</td>
<td>0.108121</td>
<td>4421205</td>
</tr>
<tr>
<td>7.txt</td>
<td>5739</td>
<td>826</td>
<td>406137</td>
<td>0.044596</td>
<td>4423548</td>
</tr>
<tr>
<td>8.txt</td>
<td>6794</td>
<td>492</td>
<td>496284</td>
<td>0.044842</td>
<td>5417161</td>
</tr>
<tr>
<td>9.txt</td>
<td>5411</td>
<td>691</td>
<td>479761</td>
<td>0.043918</td>
<td>5231887</td>
</tr>
<tr>
<td>10.txt</td>
<td>5719</td>
<td>231</td>
<td>475604</td>
<td>0.04431</td>
<td>5183158</td>
</tr>
</tbody>
</table>

Table2. The Leaderboard for the TunedIT Job Scheduling competition [21], Examined data sets: all from Table1

<table>
<thead>
<tr>
<th>Rank</th>
<th>Team</th>
<th>PreliminaryResult</th>
<th>FinalResult</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>jzbontar</td>
<td>0.043801</td>
<td>0.043878</td>
</tr>
<tr>
<td>2</td>
<td>Piotr Czerpak</td>
<td><strong>0.043801</strong></td>
<td><strong>0.043878</strong></td>
</tr>
<tr>
<td>3</td>
<td>TEAM CODES</td>
<td>0.043801</td>
<td>0.043878</td>
</tr>
<tr>
<td>4</td>
<td>Notissa</td>
<td>0.043801</td>
<td>0.043878</td>
</tr>
<tr>
<td>5</td>
<td>Jannes Verstichel</td>
<td>0.043801</td>
<td>0.043878</td>
</tr>
<tr>
<td>6</td>
<td>artem</td>
<td>0.043801</td>
<td>0.043878</td>
</tr>
<tr>
<td>7</td>
<td>podludek</td>
<td>0.043806</td>
<td>0.044216</td>
</tr>
<tr>
<td>8</td>
<td>rabitic</td>
<td>0.044403</td>
<td>0.044680</td>
</tr>
<tr>
<td>9</td>
<td>Rav</td>
<td>0.047203</td>
<td>0.046747</td>
</tr>
<tr>
<td>10</td>
<td>Baseline</td>
<td>0.197606</td>
<td>0.195016</td>
</tr>
<tr>
<td>11</td>
<td>Herald Kllapi</td>
<td>0.197606</td>
<td>0.195016</td>
</tr>
<tr>
<td>12</td>
<td>Xenopax</td>
<td>0.197606</td>
<td>0.195016</td>
</tr>
<tr>
<td>13</td>
<td>cpreston</td>
<td>0.197606</td>
<td>0.195016</td>
</tr>
<tr>
<td>14</td>
<td>Oscar</td>
<td>0.197606</td>
<td>0.195016</td>
</tr>
<tr>
<td>15</td>
<td>ga1</td>
<td>0.197606</td>
<td>0.195016</td>
</tr>
<tr>
<td>16</td>
<td>Tri Kurniawan Wijaya</td>
<td>0.197606</td>
<td>0.195016</td>
</tr>
</tbody>
</table>
4. Conclusion

Our job scheduling method is based on the best result for multi-criterion optimization reached during the aforementioned competition. It is also one of the best methods among those known for that difficult task. Our future plan is to check the effectiveness of our algorithms against the other data in the field.

Acknowledgements

This research has been supported by Grant 1309-802 from the Ministry of Science and Higher Education of the Republic of Poland.

Bibliography

Job scheduling algorithm based on multi-criteria optimization


WYKORZYSTANIE OPTYMALIZACJI WIELOKRYTERIALNEJ W ALGORYTMIE HARMONOGRAMOWANIA ZADAŃ

Streszczenie


Słowa kluczowe: harmonogramowanie zadań, optymalizacja wielokryterialna

Piotr Czerpak
Piotr Artiemjew
Wydział Matematyki i Informatyki
 Uniwersytet Warmińsko-Mazurski w Olsztynie
email: piotrczerpak@matman.uwm.edu.pl
artem@matman.uwm.edu.pl