TabuSearch: Job-shop scheduling problem

Jerzy Wroczyński (250075)

2020-06-04

1 Introduction

The job shop scheduling problem is one of many theoretic scheduling problems. In a paper by Dell'Amico and Trubian [4] it was classified as $J||C_{\max}$ using the notation introduced by Graham et al. [5]. Letter J represents „job shop scheduling problem", two vertical lines with nothing in between mean no further job characteristics are given and $C_{\max}$ defines the optimization problem as minimizing the maximum completion time of all given jobs.

Of course, there are many different types of such problems e.g. there can be a predetermined quantity of machines e.g. only one machine, jobs can have certain characteristics e.g. each job has a *fuzzy due date* etc. In this paper we will propose an algorithm for solving the problem classified as $J||C_{\max}$ and examine results of other algorithms used in other papers.

We are given following resources:

1. a set J of n jobs to schedule,
2. a set O of N atomic operations
3. a set M of m machines.

For each job J_j there is a sequence of operations $O_{i,j} \in O$ and each of these operations has to be processed without interruption separately on a machine $\mu_{i,j} \in M$ for $d_{i,j}$ units of time.

For better understanding of a such schedule problem a visual aid of a Gantt chart can be used:

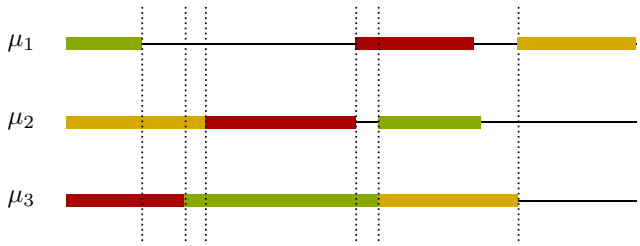

Figure 1: Example Gantt chart

which represents one feasible solution for scheduling three jobs (each consisting of three operations) on three machines. We can see now, there are some difficulties to overcome when dealing with such a problem.

For example, in this solution machine μ_1 runs idle for a very long time which can be an indicator on how good this solution is. Our main goal is to minimize the running time of the machine that completes its tasks last.

2 Problem representation

The Gantt chart very clearly shows the amount of time each task takes and for how long each machine runs idle. However, it is not a convenient way of representing this problem when it comes to applying TabuSearch.

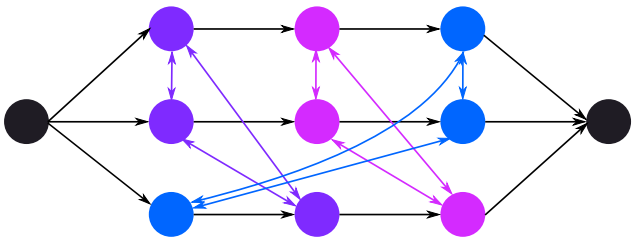

Figure 2: Example disjunctive graph

Figure 2 shows an example of a disjunctive graph that is commonly used when dealing with the job shop problem. Here, each atomic operation is represented as a node. The order of operations in each job is maintained using directed edges (black edges) drawn between appropriate operations (so they form a sequence of operations). These edges are called „conjunctive". The rest of the edges (coloured edges) are called „disjunctive" and they represent all possible combinations of a job schedule. In this abstract form we can think of them as normal, undirected edges. When a schedule is being sought these edges need to become directed as to define order of execution on each of the given machines.

When searching for a solution the original order of operations in a job has to be maintained – direction of black edges cannot be altered. Defining the direction of the undirected edges will give us a certain schedule thus a solution to our problem. One way of directing these edges is to apply an „acyclic orientation" to each group of operations executed on the same machine. Then, having created such a DAG (Directed Acyclic Graph) the longest path (with respect to the cost function) defines the quality of this solution. In other words – the goal here is to minimize the cost of this longest path in our DAG.

3 TabuSearch

The heuristic we will be applying to this problem is TabuSearch. This heuristic algorithm is an extension to LocalSearch which searches through all neighbouring solutions of the currently chosen best solution. It chooses the next solution based on the cost function. TabuSearch introduces so called „Tabu" list that holds all forbidden „moves" that cannot be performed when generating a neighbourhood of the currently chosen solution. This reduces the amount of solutions to consider and thus gives more time for greater expansion in the global solution space.

To prevent from discarding some solutions due to the Tabu list we can add an aspiration criteria. For example if a given move has been put into Tabu list a long time ago, we can again use this move when generating a new neighbourhood.

4 The proposed algorithm

Before running the TabuSearch heuristic we need to define a way of finding the first feasible solution to start from. Using the *Shortest Processing Time* rule from Brandimarte [2, section 2.1] we can generate the first feasible solution. Each time there is more than one operation waiting to be processed by a given machine the operation with the lowest processing time is scheduled for running.

For the main algorithm we are going to use the one specified by Dell'Amico and Trubian [4, section 2.]:

Algorithm 1 TabuSearch

```
1: x* ← find a feasible first solution
2: T ← ∅ ▷ the Tabu list
3: while termination criteria not met do:
4:     if T is full and there cannot be generated any more neighbouring solutions then:
5:         T ← ∅
6:     end if
7:     choose the best solution x̂ from 𝒩(x*, T)
8:     put a move which leads from x̂ to x* into T
9:     if x̂ is better than x* then:
10:        x* ← x̂
11:    end if
12: end while
```

The function $\mathcal{N}$ used in line 7 of our algorithm generates a neighbourhood of solutions with respect to the Tabu list. If no aspiration criteria is given or the aspiration criteria does not remove certain moves from the Tabu list, then the algorithm still works as it empties the Tabu list if no neighbourhood can be generated (condition in line 4).

4.1 Neighbourhood

Many different approaches to the neighbourhood structure have been proposed over the years, but for our algorithm we will use the $N1$ neighbourhood introduced by van Laarhoven et al. [8, section 3.3]. To generate a move we choose a tuple of vertices v and w, such that:

1. v and w are successive operation on some machine k
2. (v, w) is a critical edge meaning it is on the critical path (the longest path)

and alter the order in which the k-th machine processes operations v and w through swapping both of these operations.

4.2 Tabu list

Because we are using a very simple definition of the neighbourhood function, for the Tabu list we also need a very basic approach. We treat the Tabu list as a FIFO (first in, first out queue) which holds all the swaps that are forbidden to avoid unnecessary evaluation of similar solutions as to explore more neighbourhoods.

In section 5 we will determine how many items should our Tabu list hold (how old an entry has to be to be forgotten).

4.3 Example

Let us see an example of a small job shop problem and apply our approach to it.

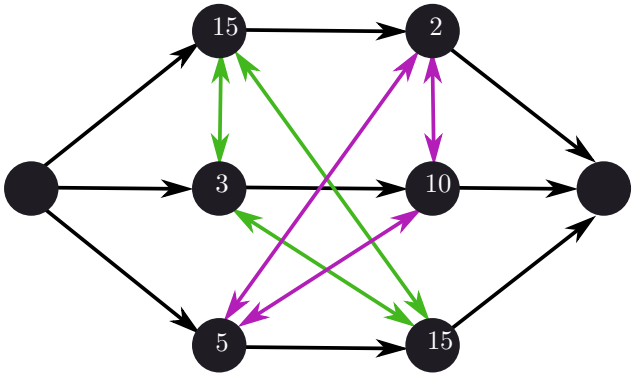

Figure 3: Exemple of a job shop scheduling problem

In figure 3 the nodes represent operations that need to be completed. The lables on the nodes represent the time needed to complete given operation (or the d parameter as defined in section 1).

Let us generate an initial solution:

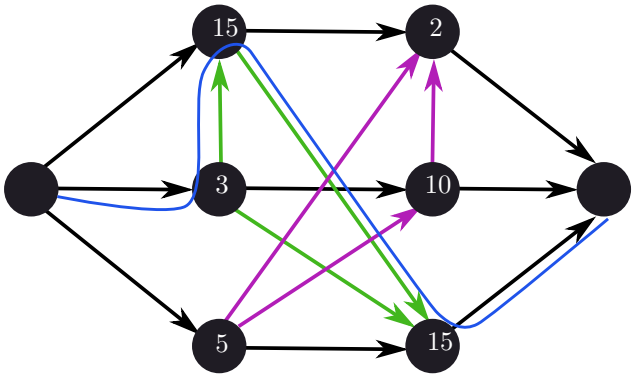

Figure 4: Example job shop scheduling problem with an initial solution applied

The blue line in figure 4 represents the longest path in this DAG.

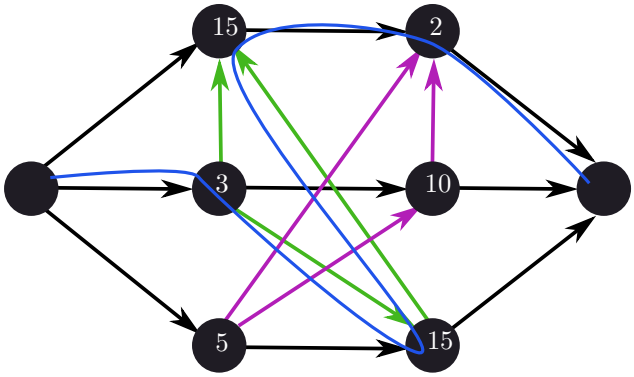

Figure 5: Example job shop scheduling problem after reversing one of (originally disjunctive) edges on the critical path

After reversing one edge as presented above in Figure 5 the cost of the longest path is higher than the initial solution generated at the beginning. Having performed a few iterations we find out that the first (initial) solution really was the best.

5 Other algorithms used in literature

We will compare the results of algorithms presented in articles by Dell'Amico and Trubian [4] and by Pezzella and Merelli [7].

Choosing an initial solution has a great impact on the final result as mentioned by Pezzella and Merelli [7] and previously observed by Dell'Amico and Trubian [4], Nowicki and Smutnicki [6].

Dell'Amico and Trubian [4] have used their own List Scheduling method called *Bidir* which creates two half-schedules that are generated starting from the beginning and from the end using the chosen priority rule. Pezzella and Merelli [7] have used a known method (proposed by Adams et al. [1]) for creating initial solutions called *shifting bottleneck* which optimizes the schedule for the machine that has the most effect on the total makespan. It solves one-machine sequencing problem for said machine.

While Dell'Amico and Trubian [4] have used a more traditional version of TabuSearch, Pezzella and Merelli [7] used a procedure by Carlier [3] for local reoptimization for the newly found better solution (based on the *shifting bottleneck* algorithm).

Dell'Amico and Trubian [4] have incorporated a restarting behaviour if no improvement was achieved after Δ iterations. Also because of empirical evidence they have discovered that introducing randomization in some aspects of the algorithm yields better computational results. For example, after Λ iterations the minimum and the maximum length of the Tabu list is randomly selected from a given range that is specified right before running the algorithm.

The algorithm by Pezzella and Merelli [7] alters the length of the Tabu list as well. It divides the maximum number of iterations into five parts. Three (odd numbered) parts keep the Tabu list length constant and during the parts numbered 2 and 4 transitions between different lengths occur.

Both approaches terminate after a certain threshold of maximum iterations (defined in both articles as *Maxiter*).

5.1 Computational results

First, let us compare the results for problems classified as *class a)* in [7] (contained in table 3 in [4]). The computational results got from the approach by Pezzella and Merelli [7] are in most cases better or the same as Dell'Amico and Trubian [4], apart from very few cases.

However, results for problems from *class b)* (table 4 in [4]) are in almost all cases better when computed using the algorithm by Pezzella and Merelli [7] than the algorithm by Dell'Amico and Trubian [4]. For example, in problem LA29 both algorithms struggle to find the optimum, but algorithm by Pezzella and Merelli [7] yields results that are at least closer to the optimum (lower error percentage).

5.1.1 Some remarks on the computational results

Unfortunately Pezzella and Merelli [7] have not included results of running their heuristic approach on generating the initial solutions alone (without the TabuSearch) so we cannot compare the results of the initial solutions generated by two approaches in the discussed papers. We can only speculate on how did these procedures have an effect on the computational results. The TSSB algorithm presented by Pezzella and Merelli [7] does include a reoptimization technique that runs after the TabuSearch finds a better solution – it may have had some impact on how quickly the algorithm converged to solutions near the optimum and the final solution (found by the TabuSearch procedure) could have been improved even more (by the *shifting bottleneck* procedure) – Dell'Amico and Trubian [4] are using only TabuSearch, their *Bidir* procedure is used exclusively for generating an initial solution.

The main difficulty that comes with applying TabuSearch is definitely prevention of cycling. Both of these approaches are trying to overcome this issue. Dell'Amico and Trubian [4] are detecting cycling by remembering some aspects of the solutions being considered in the context of *Tcycle* iterations (if the cost function is the same and the so called *witness* edge is the same a cycle is detected). Algorithm by Pezzella and Merelli [7] is altering the Tabu list length in such a way, that each block of iterations (as discussed earlier in section 5) could be associated with either expansion (finding more *good* neighbourhoods) or exploration (finding more *good* members of a given neighbourhood). By contrast, Dell'Amico and Trubian [4] approach to altering the length of the Tabu list is non-deterministic – there are values *min* and *max* which make a certain range of possible lengths. Both of these parameters are randomly selected from range $[a, b]$ and from range $[A, B]$ respectively.

References

[1] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988. ISSN 00251909, 15265501. URL `http://www.jstor.org/stable/2632051`.

[2] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Ann. Oper. Res.*, 41 (1–4):157–183, May 1993. ISSN 0254-5330.

[3] Jacques Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11 (1):42 – 47, 1982. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(82)80007-6. URL `http://www.sciencedirect.com/science/article/pii/S0377221782800076`. Third EURO IV Special Issue.

[4] Mauro Dell'Amico and Marco Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231–252, 09 1993. doi: 10.1007/BF02023076.

[5] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier, 1979. doi: https://doi.org/10.1016/S0167-5060(08)70356-X. URL `http://www.sciencedirect.com/science/article/pii/S016750600870356X`.

[6] Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996. ISSN 00251909, 15265501. URL `http://www.jstor.org/stable/2634595`.

[7] Ferdinando Pezzella and Emanuela Merelli. A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, 120(2):297 – 310, 2000. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(99)00158-7. URL `http://www.sciencedirect.com/science/article/pii/S0377221799001587`.

[8] Peter J. M. van Laarhoven, Emile H. L. Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992. doi: 10.1287/opre.40.1.113. URL `https://doi.org/10.1287/opre.40.1.113`.