Instances generation for a single machine scheduling problem

A. Lazarev, N. Pravdivets, E. Grishin, S. Galakhov
V.A. Trapeznikov Institute of Control Science RAS, Moscow.
email: galakhov.sa16@physics.msu.ru

Abstract. The branch and bound method is used to obtain exact solutions of the single machine scheduling problem with the objective function of maximum lateness minimization. Lower bounds are estimated by solving dual problems. A machine-independent value, namely the number of branching points in the search tree, is used as the complexity index of instances. The instances are presented as points in 3n-dimensional space. Known and newly developed ways of instances generation (with a large number of branching points) are presented, and the advantages and disadvantages of each of them are described.

In the process of developing methods and algorithms for solving problems, there is a need for methods for testing efficiency and stability. In general, there are two testing methods. The first one deals with real data of practical problems. However, this method has several disadvantages. It is impossible to test algorithms on real data for all possible cases (due to conditions imposed by real data). Although testing on real data can be effective for applied algorithms, it is inapplicable to testing algorithms for solving fundamental problems. To study the efficiency of the algorithms, test instances should be generated. There are test databases with test instances for fundamental problems, for example [1]. However the number of instances in these data bases is limited and not all possible cases may be taken into account. In [2] a similar problem is considered and the algorithm was tested on benchmark instances. The instances were generated by a random scheme [3]. The main disadvantage of random instance generation is that there is no guarantee that the instances will be sufficiently different from each other.

1. Problem statement
We consider the single machine scheduling problem, where a set of n jobs $N = \{1, 2, ..., n\}$ has to be processed on a single machine starting at time $\tau$. For each job $j$, a release date $r_j$, a processing time $p_j$ and a due date $d_j$ are given. Let us denote by $C_j$ the completion time of job $j$. Let's introduce the function of maximal lateness $L_{\text{max}} = \max_{j=1,...,n}\{C_j - d_j\}$. We denote by $L_{jk}(\pi)$ the value of lateness of job $j$ which is processed $k$-th in the schedule $\pi = \{j_1, j_2, ..., j_k, ..., j_n\}$. So the objective function in initial problem is minimizing the value $\mu^* = \min_{\pi \in \Pi(N)} \max_{k=1,...,n} L_{jk}(\pi)$, where $\Pi(N)$ is a set of all feasible schedules. This is the original problem $1|r_j|L_{\text{max}}$ in notation [4].

1.1 Dual problem
In the dual problem we construct the schedule with the objective function value $\nu^* = \max_{k=1,...,n} \min_{\pi \in \Pi(N)} L_{jk}(\pi)$. So we may introduce the minimal value of lateness for job which is processed by $k$-th among all feasible solutions $\Pi(N)$, $\nu_k = \min_{\pi \in \Pi(N)} L_{jk}(\pi)$. Obviously $\nu^* = \max_{k=1,...,n} \nu_k$. The dual problem is to minimize $L_{\text{max}} = \max_{j=1,...,n}\{C_j - d_j\}$ subject to the constraints $\sum_{j \in \Pi(N)} p_j \leq \tau$ and $\sum_{j \in \Pi(N)} \nu_k \leq \tau$. The algorithm for solving the dual problem is based on the branch and bound method. The lower bounds are estimated by solving dual problems. A machine-independent value, namely the number of branching points in the search tree, is used as the complexity index of instances.
\[ \max_{k=1, \ldots, n} v_k. \] In [5] it was proven that if \( \phi_j(t), j = 1, 2, \ldots, n, \) are arbitrary non-decreasing penalty functions in the problem 1\( |r_j|L_{\max} \) (in our case \( \phi_{\max} = L_{\max} \)). Then we have \( v_n \geq v_k \) for all \( k = 1, 2, \ldots, n, \) i.e. \( v^* = v_n \). So the solution of the dual problem 1\( |r_j|\phi_{\max} \) is reduced to the problem of finding the value \( v_n \). To solve this problem we enumerate all jobs in order of non-decreasing release dates: \( r_{i_1} \leq r_{i_2} \leq \cdots \leq r_{i_n} \). Due to \( v_n = \min_{\pi \in \Pi(N)} \phi_{\max}(C_j(\pi)) \) we will put each of the jobs \( j \) of the set \( N \) onto the last (i.e., the \( n \)-th) position. The other \( (n - 1) \) jobs of the set \( N \setminus \{j\} \) are arranged in their original order starting at time \( \tau \). This gives the earliest completion time of processing the jobs from the set \( N \setminus \{j\} \). This procedure is formally summarized in Algorithm 1. The input data of Algorithm 1 is the pair \((N, \tau)\), where \( \tau \) is used to calculate \( C_j \). Algorithm 1 has complexity \( O(n^2) \) operations, while original problem is NP-hard in the strong sense [6].

1.2 Branch and bound algorithm
In order to solve the initial problem 1\( |r_j|L_{\max} \) the branch and bound algorithm combined with dual problem solution was proposed [7]. In [5] the following theorem was proved. Let \( \phi_j(t), j = 1, 2, \ldots, n, \) be arbitrary non-decreasing penalty functions for the problem 1\( |r_j|\phi_{\max} \). Then the inequality \( \mu^* \geq v^* \) holds. So the dual problem could be used for lower bound estimation.

To solve the initial problem using the estimates obtained by the dual problem, we suggest the following Algorithm 2. The input data for Algorithm 2 (branch and bound method) is the set \( M = \{N', \tau', v', \pi', B'\} = \{N, \tau, v, \emptyset, \emptyset\} \), where
- \( \tau' \) is a time when the machine is ready for processing the jobs, at the beginning the machine is ready at moment \( \tau \),
- \( N' \) is a set of non-processed jobs, at the start time \( \tau \) all jobs from \( N \) are non-processed,
- \( v' \) is an estimation of lower bound which was obtained by solving the dual problem with non-processed jobs, at the start time \( \tau \) the dual problem is solved for all jobs from \( N \),
- \( \pi' \) is a constructed schedule for jobs from the set \( N' \), at the start time \( \tau \) the schedule \( \pi' = \emptyset \) because \( N = N' \),
- \( B' \) is the set of jobs that cannot be placed on the current position of the schedule according to algorithm, at the start time \( \tau \) all jobs from the set \( N \) could be placed on the first position in schedule.

Then each point (which corresponds to certain sub-instance) is replaced by two points which correspond to sub-instances. In one of these sub-instances the next job is placed on a current position in the schedule, and in the other sub-instance the job is prohibited to be in current position by including the job into set \( B \).

An estimation of the complexity of the instances will be the number of branching points in the searching tree. It is easy to evaluate the maximum number of branching points in the tree \( V(n) = n \cdot (V(n - 1) + 1) - 1 \), where \( V(1) = 0 \).
2. Instances generation

2.1 Random instances

In order to assess the performance of the proposed algorithm, it must be tested on various instances. The common way to generate test instances is random. All jobs parameters are set by random values. This method is imperfect because there is no guarantee that the different options of instances will be considered (due to the large number of variables) and the instances will not repeat themselves. Actually it is possible to use uniform distribution. It is also necessary to test the algorithm on complex instances, and a random method cannot guarantee that they will not be obtained. If the instance is presented as a point in a 3n-dimensional space (n jobs with three parameters), then it is impossible to obtain points that are sufficiently distant from the origin of the coordinates when using the uniform distribution.

In order to be able to obtain points all over the space by random generation of instances, it was suggested in [8] that the normal distribution should be used. So that there is a possibility of a sufficiently large deviation from the uniform distribution to create more 'remote' instances. Using the generated instances, a new instance was created by an algorithm and parameter variation for each job, which requires a lot of computing capacity to find the optimal solution. But in fact to cover all possible cases and space by randomly distributed instances is impossible. In Fig.1 the illustrations of such instances projection to the 3-dimensional space are presented.

2.2 Sphere L-instances

In chapter one [9], the following scheme was proposed for obtaining complex instances. A certain number of points in a 3n-dimensional sphere with a center at the origin of coordinates with a given radius are taken. Then the point with the largest complexity is chosen and a 3n-dimensional sphere is built with the center at this point (possibly with a smaller radius than the initial sphere). Then the point with the highest complexity in a new sphere is again searched for. The procedure repeats until a point is found that there are no points with higher complexity (the number of branching points in the searching tree) in the sphere built around it. This scheme also uses the generation of evenly distributed
points in the sphere. However, when looking for complex instances, it is possible to move away from the origin of the coordinates, as in the case of normal distribution. However, the difficulties related to radius lengths arise. If radiuses are low enough, we will not move far from initial point and complexity of obtained instances will be small. In opposite case if radius is too big we will obtain the similar complex point in space. So the questions connected with radiuses length are significant. In Fig. 2 the illustration of projection of Sphere L-instances to the 3-dimensional space is presented.

Fig. 1. The illustrations for random distributed instances.

Fig. 2. The illustration for sphere L-instances.

2.3 Parallelepiped P-instances

The challenge to generate instances in a 3n-dimensional sphere is difficult. It is easier to generate points in a parallelepiped with edges parallel to the axes of coordinates. The general idea of instances generation remains the same. The lengths of the sides of the initial 3n-dimensional parallelepiped are set, and evenly distributed random points are generated. The point being searched for is the one with the largest complexity. The parallelepiped is then built around the point with the coordinate \( a \) with the lengths of the sides \((a_{\text{max}} - a_{\text{min}})/2\), where \( a_{\text{max}} \) and \( a_{\text{min}} \) are the maximum and minimum values of the corresponding coordinate among all the generated points in the current parallelepiped. The procedure repeats until a point is found such that there are no points with higher complexity in the parallelepiped built around it. Unlike the L-instances, the linear dimensions of the parallelepipeds change dynamically according to the distance of the current point from the origin of the coordinates,
which has a positive effect on finding complex instances. In Fig.3 the illustration of projection of parallelepiped P-instances to the 3-dimensional space is presented.

Fig. 3. The illustration for parallelepiped P-instances.

2.4 Cube G-instances

Note that by rearranging the three coordinates \((r_i, p_i, d_i)\) and \((r_j, p_j, d_j)\) in the 3n-dimensional space of instances, the instance does not change, the optimal schedule, complexity and value of the objective function is the same for \(n!\) symmetrical points on the surface of the sphere. Therefore, we can restrict ourselves to considering only the 3-dimensional space. It should also be noted that the case where there are jobs which arrived earlier than the machine is ready is equivalent to case when all these jobs arrived when the machine is ready \((r_i = 0)\). Moreover the case when all jobs arrived after the machine is ready is identical to the case if the machine is ready at the moment when the first job arrived. Cases where the processing times are negative are not possible. Due to all the above, it is proposed to generate points on the surface of a quarter of the cube where the values of one triplet \((r_1, p_1, d_1)\) are uniformly distributed on the surface defined by the following system of inequalities:

\[
\begin{align*}
0 & \leq r_1 \leq 1, \quad 0 \leq p_1 \leq 1, \quad -1 \leq d_1 \leq 1; \\
0 & \leq r_1 \leq 1, \quad p_1 = 1, \quad -1 \leq d_1 \leq 1; \\
0 & \leq r_1 \leq 1, \quad 0 \leq p_1 \leq 1, \quad d_1 = \pm 1.
\end{align*}
\]

And the other coordinates are evenly randomly distributed at intervals of \(0 \leq r_i \leq 1, \quad 0 \leq p_i \leq 1, \quad -1 \leq d_i \leq 1, \quad i = 2, ..., n\). This approach allows checking all possible cases. Also it allows us to narrow down the area of search for instances by considering the narrowing of the 3-dimensional space area under study. In Fig.4 the illustration of projection of Cube G-instances to the 3-dimensional space is presented.

Fig. 4. The illustration for Cube G-instances.
2.5 Measure of polynomial unsolvability

Let’s represent initial instance as a vector $\vec{a}$ (from origin of coordinates to point in $3n$-dimensional space) in case of Sphere L-instances. Then we generate several initial instances which correspond to a set of vectors $\vec{a}_1, \vec{a}_2, \vec{a}_3, \ldots$, and according to proposed scheme we obtain final points $\vec{b}_1, \vec{b}_2, \vec{b}_3, \ldots$ with the highest complexity (the largest number of branching points in the searching tree). The interesting fact was determined. The initial vectors $\vec{a}_1, \vec{a}_2, \vec{a}_3, \ldots$ were located in the space independently and randomly. Instead of this the vectors $\vec{b}_1, \vec{b}_2, \vec{b}_3, \ldots$ were mostly either almost collinear (that is logical) or orthogonal (because of specific symmetry). This fact gave rise to the idea about Cube G-instances.

Note that all points (instances) lying on a line which goes through the origin of the coordinates have the same optimal schedule (but different values of the objective function). Consequently, it is possible to consider instances on the surface of a $3n$-dimensional unit sphere. There are areas of polynomially solvable cases on the surface of this sphere. As a measure of the solvability of the problem, it is suggested that the distance from a point on the surface of a sphere as far as possible from all the solvability areas to these areas is found. However, generating points on the surface of a $3n$-dimensional unit sphere is quite a complex problem. To date, the measure of the problem’s solvability $1|r_j|L_{max}$ equals to $\frac{1}{\sqrt{3}}$ [10].

3. Conclusion

This paper describes the branch and bound method to obtain exact solutions to the scheduling problem for single machine. Lower bounds are obtained by solving a dual problem. The methods of generating L-instances in a sphere, P-instances in a parallelepiped, G-instances on a cube are used to evaluate the efficiency of the algorithm. The number of branching points in the solution tree is used as a complexity index. The results of computational experiments for 9 jobs are presented in Fig. 5. From 300000 instances, which were obtained by proposed schemes, about 85% were solved really quickly. The concept of the problem solvability measure as the distance from the solvable areas to the most distant point on a unit sphere in the instance space is also described.

![Fig. 5. Number of branching points for the instances with 9 jobs.](image)

Acknowledgements

The research was partially supported by RFBR and MOST (project № 20-58-S52006).
References
[1] Shunji Tanaka Database https://sites.google.com/site/shunjitanaka/home?authuser=0
[2] Biskup D., Feldmann M., Benchmarks for scheduling on a single machine against restrictive
and unrestrictive common due dates, Computers & Operations Research, 2001, V. 28(8), 787-801.
[3] Lehmer D.H., Mathematical Methods in Large-scale Computing Units, Proceedings of the
Second Symposium on Large Scale Digital Computing Machinery, Harvard University Press, 1951,
141-146.
[4] Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Optimization and
approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics,
1979, V. 5, 287–326.
[5] Lazarev A.A., Dual of the Maximum Cost Minimization Problem., J. Sov. Math, 1989,
V. 44, 642–644.
[6] Lenstra J.K., Rinnooy Kan A.H.G., Brucker P., Complexity of Machine Scheduling
Problems, Annals of Discrete Mathematics, Elsevier, V. 1, 1977, 343–362.
[7] Lazarev A.A., Pravdivets N., Werner F., On the Dual and Inverse Problems of Scheduling
Jobs to Minimize the Maximum Penalty, Mathematics, 2020, V. 8, 1131.
[8] Hall N.G., Posner M.E., Generating Experimental Data for Computational Testing with
Machine Scheduling Applications, Operations Research, 2001, V. 49, 854–865.
[9] Lazarev A.A., Scheduling theory. Methods and algorithms, Moscow.: ICS RAS, 2019
[10] Lazarev A.A., Arkhipov D.I., Estimation of absolute error and polynomial solvability for the
classical NP-hard problem of the scheduling theory, Reports of the Academy of Sciences, 2018,
V. 480(5), 523–527.