An Exact Algorithm for Single-Machine Scheduling without Idle Time

1 author:

Shunji Tanaka
Kyoto University
50 PUBLICATIONS **629** CITATIONS

SEE PROFILE

An Exact Algorithm for Single-Machine Scheduling without Idle Time

Shunji Tanaka
Kyoto University, Kyotodaigaku-Katsura, Nishikyo-ku, Kyoto 615-8510, Japan, tanaka@kuee.kyoto-u.ac.jp

This study proposes an exact algorithm for the single-machine scheduling problem to minimize the sum of job completion costs, where machine idle time is prohibited. This algorithm is based on the the SSDP (Successive Sublimation Dynamic Programming) method, which starts from a relaxation of the original problem and then successively solves relaxations with more detailed information by dynamic programming. In this study several improvements are proposed to reduce both the memory requirement and the computational time. Numerical experiments show that the proposed algorithm can handle 300 jobs instances of the total weighted tardiness problem and 200 jobs instances of the total weighted earliness and tardiness problem.

1 Introduction

This study treats a class of scheduling problems to sequence jobs on a single machine so that the sum of job completion costs is minimized. More specifically, we consider that a set of n jobs $\mathcal{N} = \{1, \ldots, n\}$ is to be processed on a single machine without preemption. Job i ($i \in \mathcal{N}$) is given an integer processing time p_i and an integer cost function $f_i(t)$. Machine idle time is not permitted and the machine cannot process more than one job at a time. Hence, all the jobs should be processed in the interval $[0,\ T]$, where $T = \sum_{i=1}^{n} p_i$. The objective is to find an optimal job sequence to minimize $\sum_{i=1}^{n} f_i(C_i)$, where C_i denotes the completion time of job i.

For this problem, an efficient lower bound computation method based on the state-space relaxation technique [1] was proposed by Abdul-Razaq and Potts [2]. Based on their results, Ibaraki and Nakamura [3] applied the successive sublimation dynamic programming (SSDP) method [4] to this problem. In this algorithm, the constraint on successive jobs is first added to the dynamic programming state-space relaxation of the original problem (it is equivalent to the Lagrangian relaxation of the constraint on the number of job occurrences). Then, state-space modifiers are successively added until the duality gap becomes zero. In the course of the algorithm, dynamic programming states are eliminated to reduce the memory requirement by computing lower bounds for passing through the states. Nevertheless, they concluded that their algorithm is hard to apply due to the heavy memory requirement when the scheduling horizon T becomes large.

The purpose of this study is to construct an exact algorithm based on the SSDP method. There are several improvements from the original algorithm by Ibaraki and Nakamura [3]:

(1) lower bound improvement and state elimination by the dominance of adjacent jobs [5],

(2) efficient upper bound computing by the iterated enhanced dynasearch [6, 7],

(3) sophisticated step sizing in subgradient optimization,

(4) improved selection of state-space modifiers,

(5) further state elimination by the dominance of four successive jobs.

These improvements enable us to reduce both the memory requirement and the computational time, and the algorithm can handle the total weighted tardiness problem ($1||\sum w_i T_i$) instances

with up to 300 jobs, and the total weighted earliness and tardiness problem ($1||\sum(\alpha_i E_i + \beta_i T_i)$) instances with up to 200 jobs, while our implementation of the original algorithm could not solve some of the 40 jobs instances optimally.

2 Outline of the algorithm

Stage 1 The initial upper bound is computed by applying the iterated enhanced dynasearch [6, 7] to an SPT sequence. Then, subgradient optimization is applied to a Lagrangian dual corresponding to a Lagrangian relaxation of the original problem. In this relaxation, the constraint on the number of job occurrences is relaxed and the constraint that no job duplication occurs in any *two* successive jobs [2] is added. State elimination is performed by applying both forward and backward dynamic programmings every time when the lower bound is improved. After the subgradient optimization is terminated, the upper bound is updated by applying the iterated enhanced dynasearch to a solution constructed from a solution of the relaxation.

Stage 2 Starting from the multipliers obtained in Stage 1, subgradient optimization is applied to a Lagrangian dual corresponding to a more constrained relaxation of the original problem. In this relaxation, the constraint that no job duplication occurs in any *three* successive jobs [2, 3, 8] and another constraint on the dominance of adjacent jobs [5] are added. An upper bound is also computed in every five iterations, and state elimination is performed every time when either the lower bound or the upper bound is improved.

Stage 3 While state-space modifiers are successively added, the relaxation in Stage 2 is solved in forward or backward manner by turns until the current upper bound is proved to be optimal. The upper bound computation, state elimination, and further state elimination based on the dominance of four successive jobs are applied every time when the relaxation is solved. Modifiers are added to maximum two jobs separately so that they should appear exactly once. These jobs are chosen from the jobs that never appear and then that appear most frequently in the solution of the relaxation. Ties are broken by the smallest number of occurrences in the current dynamic programming state space to suppress the increase of states.

3 Numerical experiments

With regard to $1||\sum w_j T_j$, the instances in [9] are used as 40, 50 and 100 jobs instances ($n = 40, 50, 100$) (available from OR-Library: http://people.brunel.ac.uk/~mastjjb/jeb/info.html). The instances with $n = 150, 200, 250, 300$ are generated in a similar way to the OR-Library instances. Thus 125 problem instances are considered for each n. The instances of $1||\sum(\alpha_i E_i + \beta_i T_i)$ are generated from those of $1||\sum w_j T_j$, where integer earliness weights are generated by the uniform distribution in $[1, 10]$. Computation is performed on a Pentium4 2.4GHz desktop computer with 512MB RAM by running a code written in C (gcc). The maximum memory size for dynamic programming states is restricted to 384MB.

The results are shown in Table 1. Only one instance of $1||\sum(\alpha_i E_i+\beta_i T_i)$ with 200 jobs cannot be solved optimally due to the heavy memory requirement. Nevertheless, we can see that the proposed algorithm can handle 300 jobs instances of $1||\sum w_j T_j$ and 200 jobs instances of $1||\sum(\alpha_i E_i + \beta_i T_i)$. Compared to the algorithm in [10] for $1||\sum w_j T_j$ and the algorithm in [11] for $1||\sum(\alpha_i E_i + \beta_i T_i)$, our algorithm seems much faster because the algorithm in [10] requires maximum 9 hours on a Pentium4 2.8GHz computer to solve the OR-Library 100 jobs instances and the algorithm in [11] could not solve some of 40 jobs instances within 3600 seconds on a Pentium II 266MHz computer.

Table 1: Average and maximum CPU times of the proposed algorithm

n	$1\|\|\sum w_iT_i$			$1\|\|\sum(\alpha_iE_i+\beta_iT_i)$		
	CPU time (s)		solved	CPU time (s)		solved
	average	maximum		average	maximum	
40	0.20	0.99	125/125	0.25	0.52	125/125
50	0.42	0.98	125/125	0.58	1.87	125/125
100	7.16	50.04	125/125	12.01	27.69	125/125
150	30.93	213.76	125/125	68.55	285.29	125/125
200	82.21	355.60	125/125	*270.27**	*2032.58**	124/125
250	200.68	1534.78	125/125			
300	446.52	5456.41	125/125			

*The average and maximum are taken over optimally solved instances.

Moreover, the framework of this study is more general and it can be applied to any single-machine machine scheduling problems without idle time to minimize the sum of job completion costs.

Acknowledgement: This work has been partially supported by Grant-in-Aid for Young Scientists (B) 19760273, from Japan Society for the Promotion of Science (JSPS).

References

[1] N. Christofides, A. Mingozzi and P. Toth (1981), State-space relaxation procedures for the computation of bounds to routing problems, *Networks* **11**, 145 – 164.

[2] T.S. Abdul-Razaq and C.N. Potts (1988), Dynamic programming state-space relaxation for single-machine scheduling. *J. Oper. Res. Soc.* **39**, 141 – 152.

[3] T. Ibaraki and Y. Nakamura (1994), A dynamic programming method for single machine scheduling, *Eur. J. Oper. Res.* **76**, 72 – 82.

[4] T. Ibaraki, (1987), Enumerative approaches to combinatorial optimization, *Ann. Oper. Res.* **10** and **11**.

[5] S. Tanaka, S. Fujikuma and M. Araki (2006), A branch-and-bound algorithm based on Lagrangian relaxation for single-machine scheduling, *International Symposium on Scheduling 2006*, 148 – 153.

[6] R.K. Congram, C.N. Potts and S.L. van de Verde (2002), An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, *INFORMS J. Comput.* **14**, 52 – 67.

[7] A. Grosso, F. Della Croce and R. Tadei (2004), An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem *Oper. Res. Lett.* **32**, 68 – 72.

[8] L. Péridy, É. Pinson and D. Rivreau (2003), Using short-term memory to minimize the weighted number of late jobs on a single machine, *Eur. J. Oper. Res.* **148**, 591 – 603.

[9] H.A.J. Crauwels, C.N. Potts and L.N. Van Wassenhove (1998), Local search heuristics for the single machine total weighted tardiness scheduling problem, *INFORMS J. Comput.* **10**, 341 – 350.

[10] Y. Pan and L. Shi (2007), On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems, *Math. Prog. A* **110**, 543 – 559.

[11] C.-F. Liaw (1999), A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem, *Comput. Oper. Res.* **26**, 679 – 693.