Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

An exact exponential branch-and-merge algorithm for the single machine total tardiness problem

Michele Garraffa[a], Lei Shang[b], Federico Della Croce[c,d], Vincent T'kindt[b,*]

[a] DAUIN – Politecnico di Torino, Torino, Italy
[b] Laboratoire d'Informatique Fondamentale et Appliquée (EA 6300), ERL CNRS 7002 ROOT, Tours, France
[c] DIGEP – Politecnico di Torino, Torino, Italy
[d] CNR, IEIIT, Torino, Italy

A R T I C L E   I N F O

Article history:
Received 5 January 2017
Received in revised form 29 May 2018
Accepted 31 May 2018
Available online xxxx
Communicated by F.V. Fomin

Keywords:
Exact exponential algorithm
Single machine total tardiness
Branch and merge

A B S T R A C T

This paper proposes an exact exponential algorithm for the single machine total tardiness problem. It exploits the structure of a basic branch-and-reduce framework based on the well known Lawler's decomposition property that solves the problem with worst-case complexity in time $\mathcal{O}^*(3^n)$ and polynomial space. The proposed algorithm, called branch-and-merge, is an improvement of the branch-and-reduce technique with the embedding of a node merging operation. Its time complexity converges to $\mathcal{O}^*(2^n)$ keeping the space complexity polynomial. This improves upon the best-known complexity result for this problem provided by dynamic programming across the subsets with $\mathcal{O}^*(2^n)$ worst-case time and space complexity. The branch-and-merge technique is likely to be generalized to other sequencing problems with similar decomposition properties.

1. Introduction

Since the beginning of this century, the design of exact exponential algorithms for NP-hard problems has been attracting more and more researchers. Although the research in this area dates back to early 60s, the discovery of new design and analysis techniques has led to many new developments. The main motivation behind the rise of interest in this area is the study of the intrinsic complexity of NP-hard problems. In fact, since the dawn of computer science, some of these problems appeared to be solvable with a lower exponential complexity than others belonging to the same complexity class. For a survey on the most effective techniques in designing exact exponential algorithms, readers are kindly referred to Woeginger's paper [21] and to the book by Fomin and Kratsch [6].

In spite of the growing interest on exact exponential algorithms, few results are yet known on scheduling problems, see the survey of Lenté et al. [12]. Lenté et al. [11] introduced the so-called class of multiple constraint problems and showed that all problems fitting into that class could be tackled by means of the Sort & Search technique. Further, they showed that several knowns scheduling problems are part of that class. However, all these problems required assignment decisions only and none of them required the solution of a sequencing problem.

This paper focuses on a pure sequencing problem, the single machine total tardiness problem, denoted by $1||\sum T_j$. In this problem, a job set $N = \{1, 2, \ldots, n\}$ of n jobs must be scheduled on a single machine. For each job j, a processing time p_j

* Corresponding author.
E-mail addresses: michele.garraffa@polito.it (M. Garraffa), shang@univ-tours.fr (L. Shang), federico.dellacroce@polito.it (F. Della Croce), tkindt@univ-tours.fr (V. T'kindt).

https://doi.org/10.1016/j.tcs.2018.05.040
0304-3975/© 2018 Elsevier B.V. All rights reserved.

and a due date d_j are defined. The problem asks for arranging the job set into a sequence S so as to minimize $T(N,S) = \sum_{j=1}^{n} T_j = \sum_{j=1}^{n} \max\{C_j - d_j, 0\}$, where C_j is the completion time of job j. The $1||\sum T_j$ problem is NP-hard in the ordinary sense as shown by Du and Leung [3]. It has been extensively studied in the literature and many exact procedures [2,10,15,18] have been proposed. The current state-of-the-art exact method of Szwarc et al. [18] dates back to 2001 and solves to optimality instances with up to 500 jobs. The complexity of this algorithm is analyzed by Shang et al. [17]. All these procedures are search tree approaches, but dynamic programming algorithms were also considered. On the one hand, a pseudo-polynomial dynamic programming algorithm was proposed by Lawler [10] running with complexity $\mathcal{O}(n^4 \sum p_i)$. On the other hand, the standard technique of doing dynamic programming across the subsets (see, for instance, Fomin and Kratsch [6]) applies and runs with complexity $\mathcal{O}(n^2 2^n)$ both in time and in space. Latest theoretical developments for the problem, including both exact and heuristic approaches can be found in the recent survey of Koulamas [9].

In the rest of the paper, the $\mathcal{O}^*(\cdot)$ notation [21], commonly used in the context of exact exponential algorithms, is used to measure worst-case complexities. Let $T(\cdot)$ be a super-polynomial and $p(\cdot)$ be a polynomial, both on integers. In what follows, for an integer n, we express running-time bounds of the form $\mathcal{O}(p(n) \cdot T(n))$ as $\mathcal{O}^*(T(n))$. We denote by $T(n)$ the time required in the worst-case to exactly solve the considered combinatorial optimization problem of size n, i.e., the number of jobs in our context. As an example, the complexity of dynamic programming across the subsets for the total tardiness problem can be expressed as $\mathcal{O}^*(2^n)$. By the way, the number of jobs n may not be the only possible measure of the instance size. Other parameters can be chosen, based on which different complexity analysis can be conducted. For scheduling problems, some results can be found in Mnich and Wiese [14], Mnich and van Bevern [13] and Hermelin et al. [8].

To the authors' knowledge, there is no available exact algorithm for this problem running in $\mathcal{O}^*(c^n)$ (c being a constant) time and polynomial space. Admittedly, one could possibly apply a divide-and-conquer approach [1,7]. This would lead to an $\mathcal{O}^*(4^n)$ complexity in time requiring polynomial space. The aim of this work is to present an improved exact algorithm exploiting known decomposition properties of the problem. Different versions of the proposed approach are described in Section 2. A final version making use of a new technique called branch-and-merge that avoids the solution of several equivalent subinstances in the branching tree is presented in Section 3. We provide the algorithm for the worst-case scenario for the simplicity of presentation and we prove that its complexity tends to $\mathcal{O}^*(2^n)$ in time and polynomial in space. Finally, Section 4 concludes the paper with final remarks.

2. A branch-and-reduce approach

We recall here some basic properties of the total tardiness problem and introduces the notation used along the paper. Given the job set $N = \{1, 2, \ldots, n\}$, let $(1, 2, \ldots, n)$ be a LPT (Longest Processing Time first) sequence, where $i < j$ whenever $p_i > p_j$ (or $p_i = p_j$ and $d_i \le d_j$). Let also $([1], [2], \ldots, [n])$ be an EDD (Earliest Due Date first) sequence, where $i < j$ whenever $d_{[i]} < d_{[j]}$ (or $d_{[i]} = d_{[j]}$ and $p_{[i]} \le p_{[j]}$). As the cost function is a regular performance measure, we know that in an optimal solution, the jobs are processed with no interruption starting from time zero. Let B_j and A_j be the sets of jobs that precede and follow job j in an optimal sequence that is being searched. Correspondingly, $C_j = \sum_{k \in B_j} p_k + p_j$. Similarly, if job j is assigned to position k, we denote by $C_j(k)$ the corresponding completion time and by $B_j(k)$ and $A_j(k)$ the sets of predecessors and successors of j, respectively.

The main known theoretical properties are the following.

Property 1. *(Emmons [4]) Consider two jobs i and j with $p_i < p_j$. Then, i precedes j in an optimal schedule if $d_i \le \max\{d_j, C_j\}$, else j precedes i in an optimal schedule if $d_i + p_i > C_j$.*

Property 2. *(Lawler [10]) Let job 1 in LPT order correspond to job $[k]$ in EDD order. Then, job 1 can be set only in positions $h \ge k$ and the jobs preceding and following job 1 are uniquely determined as $B_1(h) = \{[1], [2], \ldots, [k-1], [k+1], \ldots, [h]\}$ and $A_1(h) = \{[h+1], \ldots, [n]\}$.*

Property 3. *(Szwarc and Mukhopadhyay [19]) For any pair of adjacent positions $(i, i+1)$ that can be assigned to job 1, at least one of them can be eliminated.*

In terms of complexity analysis, we recall (see, for instance, Eppstein [5]) that, if it is possible to bound above $T(n)$ by a recurrence expression of the type $T(n) \le \sum_{i=1}^{h} T(n - r_i) + \mathcal{O}(p(n))$, then we have $\sum_{i=1}^{h} T(n - r_i) + \mathcal{O}(p(n)) = \mathcal{O}^*(\alpha(r_1, \ldots, r_h)^n)$ where $\alpha(r_1, \ldots, r_h)$ is the largest root of the function $f(x) = 1 - \sum_{i=1}^{h} x^{-r_i}$.

A basic branch-and-reduce algorithm TTBR1 (Total Tardiness Branch-and-Reduce version 1) can be designed by exploiting Property 2, which allows to decompose the problem instance into two smaller subinstances when the position of the longest job l is given. The basic idea is to iteratively branch by assigning job l to every eligible branching position and correspondingly decompose the instance. Each time job l is assigned to a certain position i, two different subinstances are generated, corresponding to schedule the jobs before l (inducing subinstance $B_l(i)$) or after l (inducing subinstance $A_l(i)$), respectively. The algorithm operates by applying to any given job set S starting at time t function $TTBR1(S,t)$ that computes the corresponding optimal solution. With this notation, the original instance is indicated by $N = \{1, \ldots, n\}$ and the optimal solution is reached when function $TTBR1(N, 0)$ is computed.