Theoretical Computer Science 540–541 (2014) 13–26

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Reoptimization in machine scheduling

Nicolas Boria [a,1], Federico Della Croce [b,c,*]

[a] Dalle Molle Institute for Artificial Intelligence (IDSIA), Manno, Switzerland
[b] D.A.I., Politecnico di Torino, Italy
[c] CNR, IEIIT, Torino, Italy

ARTICLE INFO

Article history:
Received 26 September 2012
Received in revised form 26 February 2013
Accepted 2 April 2014

Keywords:
Reoptimization
Machine scheduling
Min-sum cost function

ABSTRACT

This paper studies reoptimization versions of various min-sum scheduling problems. The reoptimization setting can generally be described as follows: given an instance of the problem for which an optimal solution is provided and given some local perturbations on that instance, we search for a near-optimal solution for the modified instance requiring very little computation. We focus on two kinds of modifications: job-insertions and job-deletions. For all reoptimization problems considered, we show how very simple reoptimization algorithms can ensure constant approximation ratios, and also provide some lower bounds for whole classes of reoptimization algorithms.

1. Introduction

In this paper, we present approximation algorithms for various min-sum scheduling problems in the reoptimization setting, which can be described as follows: considering an instance I of a given problem Π for which an optimal solution OPT is provided, and an instance I' which results from a local perturbation of I, can the information provided by OPT be used to solve I' in a more efficient way (i.e., with a lower complexity and/or with a better approximation ratio) than if this information was not available?

The first mention of the reoptimization framework can be found in [25] regarding a whole class of scheduling problems and the setting was formally defined in [2] for METRIC TSP. Since then, many other optimization problems have been discussed in this setting, including STEINER TREE [5,7,9,10,12,20,28], MINIMUM SPANNING TREE [15], as well as various versions of TSP [3,4,8,11]. In [16,17] positive and negative approximation results are provided for various hereditary problems including MAX INDEPENDENT SET, MAX k-COLORABLE SUBGRAPH, and MAX PLANAR SUBGRAPH in the reoptimization setting where the vertex set is modified. In [6], the MAX INDEPENDENT SET problem, as well as MIN VERTEX COVER and MIN SET COVER problems, is discussed in a similar fashion although that perturbations there concern the edge-set of the initial graph. Alternative reoptimization settings, where multiple solutions are given for the initial instance are also introduced and discussed in [13,14]. The interested reader can find a survey of all these results in [18]. A recent PhD dissertation on reoptimization can be found in [27].

Although the first results regarding reoptimization were provided on a parallel machines scheduling problem with forbidden sets in [25], most problems tackled afterwards in this setting have been graph problems and to the authors' knowledge, no other scheduling problem has been discussed in this setting since then.

* Corresponding author.
E-mail addresses: nicolas.boria@supsi.ch (N. Boria), federico.dellacroce@polito.it (F. Della Croce).

[1] Research supported by the Swiss National Science Foundation project 200020_144491/1 "Approximation Algorithms for Machine Scheduling Through Theory and Experiments".

http://dx.doi.org/10.1016/j.tcs.2014.04.004
0304-3975/

This is however somehow in contrast with the important literature available on reactive scheduling (see [26]). As mentioned for instance in [21], reactive scheduling refers to the schedule modifications that may have to be made during project execution starting from a baseline schedule. In some cases, the reactive scheduling effort may rely on very simple techniques aimed at a quick schedule consistency restoration. To this purpose, by considering an optimal schedule as baseline schedule and some jobs additions or deletions as schedule disruptions, we can see that scheduling reoptimization may be considered as strictly linked to reactive scheduling, particularly with simple reoptimization strategies so that the baseline schedule is only mildly modified.

This work tackles the reoptimization versions of various strongly NP-hard min-sum scheduling problems, under modification of the job set (insertion or deletion of jobs). We analyze the approximation ratios of simple reoptimization strategies, that merely insert the new job (jobs) into the initial optimum under job insertion, or keeps the scheduling unchanged under job deletion. Hence, all reoptimization strategies presented require at most $O(n)$ time. Yet, despite their extreme simplicity, we show that all these strategies ensure constant approximation ratios, and also provide tight lower bounds for some of them. We focus first on the total completion time performance measure and we analyze the performance of simple and natural reoptimization strategies. In particular, we study the single machine total completion time scheduling problem with different release times (denoted as $1|r_j|\sum C_j$ according to the three-field notation of [22]), the parallel machine total completion time scheduling problem with different release times (denoted as $P|r_j|\sum C_j$) and the total completion time permutation flow shop problem with different release times (denoted as $F|r_j, perm|\sum C_j$) both under job insertion and deletion. Then, we consider the reoptimization version of the problem of maximizing the weighted sum of early jobs with different release times in a parallel machines setting herewith denoted as $P|r_j|\max\sum w_j\bar{U}_j$ (which is equivalent to the more commonly considered $P|r_j|\sum w_jU_j$ problem) both under job insertion and deletion. Note that, as all these problems are strongly NP-hard, the same obviously holds for the corresponding reoptimization version under job insertion. Indeed, if one of these reoptimization versions were polynomial or pseudo-polynomial, then, starting from a trivial instance (such as an empty instance, or an instance with a single job), it would suffice to incrementally build an optimal solution for any instance, adding jobs one at a time and running the polynomial or pseudo-polynomial optimal algorithm after each insertion. The whole procedure would then be polynomial or pseudo-polynomial and would produce an optimal solution for any instance of the problem considered. On the other hand, as far as the reoptimization version of these problems under job deletion is concerned, a specific proof of NP-hardness in the strong sense is provided for each problem considered.

Notice that, with respect to classical approximation, a ratio of $\frac{e}{e-1}\approx 1.58$ was given in [19] for the $1|r_j|\sum C_j$ problem and a ratio of $O(\min\{m^{1/2}, n^{1/2}\})$ was given in [24] for the $F|r_j, perm|\sum w_jC_j$ problem. Also, polynomial time approximation schemes have been proposed in [1] for the $1|r_j|\sum C_j$ problem and for the $P|r_j|\sum w_jC_j$ problem, although here at the expense of a considerable computational effort when the deviation ϵ allowed is limited.

2. Minimizing total completion time under job insertion

In this section, we analyze the approximability of three min-sum scheduling problems, in the reoptimization setting where a single job denoted by x is inserted in an instance for which an optimal scheduling $\mathrm{OPT} = [1, \ldots, n]$ is already known.

For all problems, we will analyze a set of very simple and natural reoptimization strategies, that we will denote by $\mathtt{A}_{i,j}$. For a given couple of integers i, j, $1 \leqslant i \leqslant j \leqslant n+1$, the reoptimization algorithm $\mathtt{A}_{i,j}$ produces $(j-i+1)$ solutions, that consist of inserting the new job in the initial optimal sequence between the kth and the $(k+1)$th for all possible k's between i and j, while leaving the rest of the scheduling unchanged. Then the algorithm outputs the best solutions among these $j-i+1$ candidates. For example the algorithm $\mathtt{A}_{1,n+1}$ will test all $n+1$ possible positions (before job 1, between jobs 1 and 2, and so on up to the last possible position, which is after job n) to insert the new job x in the initial scheduling.

Remark 1. For two couples of integers i, j and i', j', such that $i' \leqslant i$ and $j' \geqslant j$, the algorithm $\mathtt{A}_{i',j'}$ always computes a solution that is at least as good as $\mathtt{A}_{i,j}$, since all candidate solutions tested by $\mathtt{A}_{i,j}$ are also tested by $\mathtt{A}_{i',j'}$.

Here, we focus specifically on algorithm $\mathtt{A}_{n,n+1}$, that produces only two solutions: it computes a first solution SOL_1 by placing the job x in the very last position, while keeping the rest of the initial scheduling unchanged, and a second solution SOL_2 by positioning x in the penultimate position, leaving the rest of the scheduling unchanged (hence, jobs $(1, \ldots, n-1)$ are before job x, and n follows x in the last position), and returning the best solution SOL between SOL_1 and SOL_2.

Note that this algorithm does not only provide interesting approximation results as we will see below, but, considering that it always schedules the new job towards the end of the initial solution, it also has the advantage of being implementable even in on-line settings where the new job is inserted after the scheduling process has started. Moreover, we show that testing more positions does not guarantee a dramatic approximation improvement in the worst case.

We present tight approximation results for the algorithm $\mathtt{A}_{n,n+1}$, and finally provide a general negative result for the whole family $\mathtt{A}_{i,j}$.