Approximability of Scheduling with Fixed Jobs *

Mark Scharbrodt † Angelika Steger ‡ Horst Weisser †

Abstract

We investigate a variant of the standard scheduling problem of minimizing makespan, where some jobs are already fixed in the schedule. The remaining jobs are to be assigned to the machines in such a way that the makespan is minimized. This problem variant (called *scheduling problem with fixed jobs*) is the root of many real world scheduling problems where machine down times and/or pre–assignments of jobs on the machines are considered. We present a polynomial time approximation scheme ($PTAS$) for the constant machine case of the scheduling problem with fixed jobs. Moreover, we will demonstrate that, assuming $P \neq NP$, there is no arbitrarily close approximation in the general case when the number of machines is part of the input. We finally show that there exists no $FPTAS$ in the constant machine case, unless $P = NP$.

1 Introduction.

In the standard scheduling problem of minimizing the makespan one is given a set J of n independent jobs with processing times p_j to be scheduled on m identical machines. The optimization problem is to assign all jobs to the machines, minimizing the completion time (the makespan) C_{max} of the last job that finishes. Even though this problem is NP–hard, it is usually considered as a relatively easy problem, in the sense that quite good and efficient approximation algorithms exist (see especially [2] and [3]).

Unfortunately, the simple set–up of the scheduling problem of minimizing makespan that the machines are freely available until all jobs are processed strongly conflicts with many real world applications. Consider for example applications in production planning. Here, the schedulers are confronted with black out times on the machines due to maintenance services or unavailability of staff (evenings, weekends). Also, some of the jobs may have been manually planned before any scheduling run is carried out. Such a situation is often found in interactive planning where the planning staff assumes a manual control on the creation of the schedule.

Modeling such kind of scheduling problems is easily achieved by considering some of the jobs, say k of the n jobs, as *fixed*. That is, they are a priori assigned to be performed on a given machine at a given time. The objective is to schedule the remaining $n-k$ jobs in such a way that C_{max} is minimized. Using a straightforward reduction from $PARTITION$ one immediately checks that the scheduling problem with fixed jobs is already NP–hard in the single machine case. Scheduling with fixed jobs has appeared sporadically under various names in the literature [4], but to our knowledge the approximability has not been considered yet. We will characterize the approximability of the scheduling problem with fixed jobs by the following three theorems:

THEOREM 1.1. *For every fixed $\epsilon > 0$ there exists an algorithm A_ϵ which finds in time $\mathcal{O}(n \log n)$ a feasible schedule for an instance I of the scheduling problem with fixed jobs and a constant number of machines. The performance ratio of algorithm A_ϵ is $1+\epsilon$.*

THEOREM 1.2. *For every $m \geq 1q$ there is no fully polynomial time approximation scheme for the scheduling problem with fixed jobs on m machines, unless $P = NP$.*

THEOREM 1.3. *If the number m of the machines is part of the input, there exists no polynomial time algorithm that solves the scheduling problem with fixed jobs within a performance guarantee of $\frac{5}{4} - \epsilon$ for all $\epsilon > 0$, unless $P = NP$.*

Theorem 1.2 and Theorem 1.3 can be proven by reduction from 3–$PARTITION$. For Theorem 1.1 we describe an algorithm A_ϵ for a fixed $\epsilon > 0$ and a constant number m of machines. The proof that algorithm A_ϵ has the desired properties is deferred to the full paper.

2 The algorithm A_ϵ.

Algorithm A_ϵ proposes subsequently candidates C for the makespan within a binary search procedure. Here we transform the scheduling problem into a packing problem with variable sized bins (shortly called VBP): For each such candidate C the gaps between two subsequent fixed jobs on the same machine and before the

*A full version of our paper is available via *http://www.informatik.tu-muenchen.de/~steger* .

†Lehrstuhl für Brauereianlagen und Lebensmittel–Verpackungstechnik, Technische Universität München in Weihenstephan, D–85350 Freising–Weihenstephan, Germany.

‡Institut für Informatik, Technische Universität München, D–80290 München, Germany.

first fixed job on every machine define "closed" bins which must not be overloaded. Similarly, the remaining extra gap on every machine that starts after the last fixed job and ends at C defines an "open" bin which may be overloaded. The items to be packed are the $n-k$ jobs that are not fixed. Obviously, C is an upper bound for the makespan if and only if all items can be packed into the bins. The fact that closed bins must not be overloaded imply that the classical approaches for approximately solving bin packing instances are not applicable directly. Approaches that round items sizes must now carefully observe the given bin sizes. In addition, small items are also hard to pack if there exist many bins that are equally small. The core of our algorithm for detecting a possible feasible packing for the problem VBP is thus a partition of the problem VBP into a family of (reasonably) unrelated restricted packing problems ($RVBP$). This is embraced by a series of greedy packings for items not considered in the $RVBP$s.

2.1 The partition procedure. The following lemma allows us to partition the packing problem VBP into separate instances of packing problems in each of which the minimum and the maximum bin capacity will not differ "too much".

LEMMA 2.1. *Given an instance I for the packing problem VBP defined via C where the bins are sorted according to non–increasing sizes we can, for $t = \lceil \frac{4}{\epsilon} \rceil + 3$ and $\epsilon < 1$, identify in linear time a partition $B = \hat{B} \cup B_1 \cup \ldots \cup B_r$ of the bin set B such that $\max(\hat{B}) \leq \epsilon \cdot \max(B)$ and $SIZE(\hat{B}) \leq \epsilon \cdot SIZE(B)$, and such that for all $i = 1, \ldots, r$: $\min(B_i) \geq \epsilon^t \max(B_i)$, and $\max(B_{i+1}) < \epsilon \min(B_i)$. Here, $\max(B)$ and $SIZE(B)$ resp. denote the size of the largest bin contained in B and the total size of all bins contained in B.*

Lemma 2.1 shows that the bins of $\hat{B}$ have no significant impact on the makespan and so they will be just packed in a greedy fashion. The remaining bin classes B_i define (restricted) variable sized bin packing problems $RVBP_i$ with bin set B_i and item set $J_i := \{p \in J \mid \epsilon \min(B_i) \leq size(p) \leq \max(B_i)\}$. Note, that by simply scaling all bin and item sizes by $\frac{1}{\max(B_i)}$ all packing problems $RVBP_i$ have the property that the size of the maximum item resp. bin is at most 1 and that the minimum item resp. bin is at least ϵ^{t+1} resp. ϵ^t. Thus the problems $RVBP_i$ can all be viewed as a single problem type which we call $RVBP$.

2.2 Solving a problem $RVBP$. The problem $RVBP$ has been constructed in such a way that all items are reasonably large in size. We now extend the technique of *linear grouping* (see [1]) and simultaneously operate on items *and* bins in order to obtain a problem where the number of items and bins takes on only a few distinct values. The resulting packing problem can be solved as an integer linear program. It will turn out that linear grouping affects the value of the optimal solution only marginally, except for problem $RVBP_1$ where the size of the items to be considered might be too large for a close approximation of the makespan. However, problem $RVBP_1$ can be solved in constant time by complete enumeration in the case that the number of items with size at least ϵ^{t+2} does not exceed $\frac{m}{\epsilon^{t+2}}$ (recall that m is fixed !). Otherwise, we can safely infer that there exists no feasible packing for VBP.

2.3 Greedy packing. As a final ingredient of our algorithm a greedy procedure will pack all the items that have not been assigned to any bin yet. Those are the items that remain unpacked in the solution of the problem $RVBP_i$ together with the items that have not been considered in any packing subproblem $RVBP_i$, as simply their sizes are "between" the item sizes of two subsequent packing problems. We proceed as follows: After having solved a problem $RVBP_i$ ($i = 1, \ldots, r$) we pack the items that remain unpacked in the solution of the $RVBP_i$ into the bins with size greater than $\max(B_i)$ (including bins from $\hat{B}$). Secondly, we greedy pack for each i all the items p which have a size that is less than the size of the smallest item contained in J_i, but larger than the largest item contained in J_{i+1}. In the greedy packing, we allow the bins, even the "closed" bins, to be overloaded by an item p – if its size is not larger than that of the bin. If items remain unpacked, there exists no feasible packing. At the end of the algorithm we remove all items that overload a closed bin and pack them with a greedy algorithm into the open bins. This final step will guarantee that the corresponding schedule will be feasible.

References

[1] W. Fernandez de la Vega and G. S. Luecker, *Bin packing can be solved within $1+\epsilon$ in linear time*, Combinatorica ,1 (1981), pp. 349–355.

[2] D. K. Friesen, *Tighter bounds for the multifit processor scheduling algorithm*, SIAM J. Comput., 13 (1984), pp. 170–181.

[3] D. S. Hochbaum and D. B. Shmoys, *Using dual approximation algorithms for scheduling problems: practical and theoretical results*, Journal of ACM, 34(1) (1987),pp. 144–162.

[4] G. J. Schmidt, *Scheduling on semi-identical processors*, ZOR-Zeitschrift für Operations Research, 28 (1984), pp. 153–162.