Operations Research Letters 39 (2011) 359–362

Contents lists available at SciVerse ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

A 3/2-approximation algorithm for k_i-partitioning

Hans Kellerer [a,*], Vladimir Kotov [b]

[a] Institut für Statistik und Operations Research, Universität Graz, Universitätsstraße 15, A–8010 Graz, Austria

[b] Belarussian State University, Faculty of Applied Mathematics and Computer Science, Prospekt Nezavisimosty 4, 220030 Minsk, Belarus

ARTICLE INFO

Article history:
Received 19 November 2010
Accepted 15 May 2011
Available online 13 June 2011

Keywords:
Scheduling
Parallel machines
Partition problem
Worst-case analysis

ABSTRACT

We consider the multiprocessor scheduling problem with the objective of minimizing the makespan such that the number of items on each machine does not exceed a machine dependent cardinality limit. We present an elementary approximation algorithm with worst-case performance ratio 3/2 and running time linear in the number of items.

1. Introduction

In this work we address the problem of assigning n independent items to m parallel machines, each having an associated machine dependent cardinality limit, such that the maximum completion time is minimized and the number of items on each machine does not exceed the corresponding cardinality limit.

More formally, we are given a set $N = \{1, 2, \ldots, n\}$ of n independent items with processing times or lengths $p_1, \ldots, p_n$. Each item has to be assigned to one of m parallel machines $M_1, M_2, \ldots, M_m$. Each machine M_i has a *cardinality limit* or *capacity* $c(i) = k_i$, i.e. at most k_i items can be processed on machine M_i with $n \leq \sum_{i=1}^{m} k_i$. The objective is to minimize the makespan. The problem is a generalization of the classical parallel machine problem $P|\ \cdot\ |C_{\max}$, i.e. we have a parallel machine problem with limited number of items per machine. Since the problem corresponds to partitioning a set of integers into a given number of subsets, it is called the *k_i-partitioning problem* (k_i-PP) as introduced in [1].

The k_i-PP is a generalization of the classical multiprocessor scheduling problem $P|\ \cdot\ |C_{\max}$ of assigning independent jobs to parallel, identical machines with the objective of minimizing the makespan. Since $P|\ \cdot\ |C_{\max}$ is well-known to be strongly NP-hard, the same holds for k_i-PP.

The fact that k_i-PP is NP-hard calls for design and analysis of approximation algorithms. Recall that a heuristic for a minimization problem that finds for any instance a solution such that the value of the heuristic solution is at most $\rho \geq 1$ times the optimal solution value is called a *ρ-approximation* algorithm; the value of ρ is called a *worst-case performance ratio*. Given an $\varepsilon > 0$, a family of $(1+\varepsilon)$-algorithms is called a *polynomial-time approximation scheme (PTAS)* if its running time depends polynomially on the length of the encoded input.

If the cardinality limits of the machines are identical, i.e. $c(i) = k,\ i = 1, \ldots, m$, the corresponding problem is called the *k-partitioning problem* (k-PP). An approximation algorithm with performance ratio 4/3 for problem k-PP was presented by Babel et al. [1]. Upper and lower bounds for k-PP have been developed by Dell'Amico and Martello [5] and Dell'Amico et al. [3]. Lower bounds and heuristic algorithms for k_i-PP have been presented by Dell'Amico et al. [4]. The paper by Chi et al. [2] contains a 3-approximation for k_i-PP. Recently, Saha and Srinavasan [6] developed a rounding method which yields a generalization of the work of Shmoys and Tardos on the generalized assignment problem [7] and yields a 2-approximation for k_i-PP as a special case.

In this work we present an elementary approximation algorithm for k_i-PP with worst-case performance ratio 3/2. The algorithm is based on guessing the number of items greater than half of the optimal makespan, and assigning the items blockwise to the machines in ascending order of capacities. This guarantees that the schedule obtained has a makespan which is not larger than 3/2 times an appropriate lower bound. The running time of the algorithm is linear in the number of items.

Let $I \subseteq N$ be a set of items. Then, $p(I)$ denotes the total processing time of items in I. Let M be a machine to which a set of items I has been assigned. Then, $\ell(M) := p(I)$ denotes the *load* of machine M. As usual, C^* denotes the makespan of an optimal schedule and C^H the makespan of a heuristic schedule.

* Corresponding author.
E-mail addresses: hans.kellerer@uni-graz.at (H. Kellerer), kotovVM@yandex.ru (V. Kotov).

0167-6377/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.orl.2011.06.005

W.l.o.g., let the items be sorted in non-increasing order of processing times, i.e. $p_1 \geq p_2 \geq \cdots \geq p_n$, and the machines be sorted in non-decreasing order of capacities, i.e. $k_1 \leq k_2 \leq \cdots \leq k_m$. By adding dummy items of length zero we may assume that $\sum_{i=1}^{m} k_i = n$, i.e. exactly k_i items have to be processed on each machine.

The work is divided into sections as follows. In Section 2 a lower bound is constructed which will be used for the algorithm presented in the next section. In Section 3 we describe the 3/2-approximation algorithm and its worst-case analysis.

2. A lower bound dependent on the items greater than $C^*/2$

We call items greater than $C^*/2$ *big items*; items not greater than $C^*/2$ are called *small items*. Let $\ell \in \{0, 1, \ldots, m\}$ be the number of big items. Due to the sorting of the items the big items are exactly the items $1, \ldots, \ell$.

For $h_2 = 1, \ldots, \ell$, $h_1 = 1, \ldots, h_2+1$, $v = 0, \ldots, m-\ell$ denote with

$$c(h_1, h_2; v) := \sum_{j=h_1}^{h_2} k_j + \sum_{j=m-v+1}^{m} k_j$$

the total capacity of the machines $M_{h_1}, M_{h_1+1}, \ldots, M_{h_2}, M_{m-v+1}, M_{m-v+2}, \ldots, M_m$.

Set $h := h_2 - h_1 + 1$ and define for $h + v > 0$

$$LB(h_1, h_2; v) := \frac{1}{h+v}\left(\sum_{j=h_1}^{h_2} p_j + \sum_{j=n-c(h_1,h_2;v)+h+1}^{n} p_j\right). \tag{1}$$

Finally, we set

$$LB := \max\{\max_{h_1,h_2,v} LB(h_1, h_2; v), 2p_{\ell+1}\}. \tag{2}$$

Lemma 1. *The value LB defined in (2) is a lower bound for the optimal solution value C^*.*

Proof. Since item ℓ is the smallest big item, we have $2p_{\ell+1} \leq C^*$. Let h_1, h_2, v be given. The other bound can be obtained as follows. At most $h_1 - 1$ big items can be put on the machines $M_1, \ldots, M_{h_1-1}$. Thus, there are h machines among the machines $M_{h_1}, M_{h_1+1}, \ldots, M_m$ which contain in total at least h of the big items $\{p_1, \ldots, p_{h_2}\}$ in the optimal solution. Denote the set of these h machines by $\mathcal{M}'$.

Choose v machines from $M_{h_1}, M_{h_1+1}, \ldots, M_m$ with largest capacities and which are not in $\mathcal{M}'$. Together with $\mathcal{M}'$ we obtain a set $\mathcal{M}$ of $h + v$ machines. The total length of the big items in $\mathcal{M}$ is at least $\sum_{j=h_1}^{h_2} p_j$ which corresponds to the first term in (1). The total number of the remaining items assigned to $\mathcal{M}$ is at least $c(h_1, h_2; v) - h$. Their total length is at least the sum of the $c(h_1, h_2; v) - h$ smallest items which corresponds to the second term in (1). An averaging argument gives $C^* \geq LB(h_1, h_2; v)$. By taking the maximum over all possible values for $LB(h_1, h_2; v)$ we get that LB is a lower bound for the optimal solution value. □

3. A 3/2-approximation algorithm

In this section we present a 3/2-approximation algorithm for k_i-partitioning. Recall that items greater than $C^*/2$ are called *big items*; the other items are called *small items*. This algorithm guesses the number of big items ℓ by running a procedure $P(\lambda)$ for $\lambda = 0, 1, \ldots, m$ and choosing the best solution. The procedure $P(\lambda)$ assigns the items $p_1, \ldots, p_\lambda$ in decreasing order to the machines $M_1, \ldots, M_\lambda$. Then, machines are filled in m iterations in increasing order, i.e., first, items are assigned to machine M_1, then, to machine M_2, and so on. Let the list $L := (\lambda + 1, \lambda + 2, \ldots, n)$ contain the remaining items. The list is updated by removing those items which have been assigned to machine M_i in iteration i.

For describing the algorithm more formally, we introduce some further notation. Define L_i to be the list of non-assigned items sorted in non-increasing order after iteration i and $A_i := L \setminus L_i$ the list of assigned items. Set $L_0 := L$ and $A_0 := \emptyset$.

For $i = 1, \ldots, m$ set

$$q_i := \begin{cases} p_i, & i = 1, \ldots, \lambda, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$\tilde{k}_i := \begin{cases} k_i - 1, & i = 1, \ldots, \lambda, \\ k_i, & \text{otherwise.} \end{cases}$$

For $1 \leq m$, let s_i denote the last (smallest) element in A_i. Set $s_0 := \lambda$. Hence, s_{i-1} is the item with smallest processing time which has been assigned to a machine before iteration i, $i \geq 1$.

If L_{i-1} contains at least $\tilde{k}_i$ items with index greater than v, then let $T_i(v)$ consist of the $\tilde{k}_i$ consecutive items in L_{i-1} starting with $v + 1$. If there are less than $\tilde{k}_i$ items with index greater than v in L_{i-1}, the set $T_i(v)$ will consist of the last (smallest) $\tilde{k}_i$ items of L_{i-1}.

Then, procedure $P(\lambda)$ and algorithm A can be described as follows:

Procedure $P(\lambda)$:

1. Assignment of the biggest λ items:
   For $i := 1$ to λ assign item i to machine M_i.
2. Assignment of the remaining items:
   For $i := 1$ to m do the following:
   2.1 If $LB \leq q_i + p(T_i(s_{i-1})) \leq \frac{3}{2}LB$: Assign items of $T_i(s_{i-1})$ to machine M_i.
   2.2 If $q_i + p(T_i(s_{i_1})) > \frac{3}{2}LB$: If there is a smallest index $t > s_{i-1}$ such that $q_i + p(T_i(t)) \leq \frac{3}{2}LB$, assign items of $T_i(t)$ to machine M_i. Otherwise, assign the last (smallest) $\tilde{k}_i$ items of L_{i-1} to M_i.
   2.3 If $q_i + p(T_i(s_{i-1})) < LB$: If there is a largest index $r < s_{i-1}$ such that $q_i + p(T_i(r)) \geq LB$, assign items of $T_i(r)$ to machine M_i. Otherwise, assign the first (largest) $\tilde{k}_i$ items of L_{i-1} to M_i.

Algorithm A:

1. For $\lambda := 0$ to m run procedure $P(\lambda)$.
2. Output the best solution.

Recall that ℓ is the number of items greater than $C^*/2$, $0 \leq \ell \leq m$. We will show in the following that $P(\ell)$ returns a solution makespan not exceeding $\frac{3}{2}C^*$. Notice that $p_\ell > C^*/2 \geq LB/2$, $p_{\ell+1} \leq LB/2$ and $p_1 \leq LB$ where the last inequality follows from the bound $LB(1, 1; 0)$.

A *block B* is a maximal subset of items of the list of assigned items A_i such that all elements of B are consecutive in L. Every list A_i is uniquely partitioned into disjoint blocks. The blocks will be numbered in increasing order, i.e. $A_i = (B_i(1), B_i(2), \ldots)$. Given two items i_1, i_2 with $i_1 < i_2$, then from $i_1 \in B_i(j_1)$, $i_2 \in B_i(j_2)$ it follows that $j_1 \leq j_2$. Moreover, denote with $B_i(la)$ the last block of A_i.

The properties of the algorithm are described in several observations.

Observation 1. *Let $B_i(v)$ be a block of iteration i. Then, there is a block $B_{i+1}(w)$ in iteration $i + 1$ such that $B_i(v) \subseteq B_{i+1}(w)$.*

Proof. Due to the maximality of a block, the elements of $B_i(v)$ are not distributed to several blocks in later iterations $j > i$. □

The next observation is obvious:

Observation 2. *Let $B_i(i_1), B_i(i_2)$ be two blocks with $i_1 < i_2$. If $B_i(i_1) \subseteq B_j(j_1)$ and $B_i(i_2) \subseteq B_j(j_2)$ for some iteration $j > i$, then $j_1 \leq j_2$.*

Let S_i denote the set of small items assigned to machine M_i in iteration i of step 2 of procedure $P(\ell)$. We say that a machine M_i is *associated with block B* if $S_i \subseteq B$.