Yokohama Publishers
ISSN 1349-8169 ONLINE JOURNAL

Pacific Journal of Optimization
Vol. 10, No. 3, pp. 577–591, July 2014

A COMBINATION OF PARALLEL MACHINE SCHEDULING AND THE COVERING PROBLEM

Zhenbo Wang*, Wenyi Hong and Dong He

Abstract: This paper studies a combination problem of parallel machine scheduling and the covering problem. We say a combinatorial optimization problem is a covering problem if it minimizes a linear objective function under some constraints and the variables are binary. The covering problem generalizes many classic combinatorial optimization problems, e.g. the shortest path problem, the vertex cover problem and the hitting set problem. The combination problem considered in this paper is to assign some of the jobs on m parallel machines such that the jobs correspond to a feasible solution of the covering problem and the objective is to minimize the makespan. We propose a unified approximation algorithm for the combination problem in which the parallel machines are uniformly related. An improved result is obtained when the parallel machines are identical. We apply our results to some specific combination problems to demonstrate our approach.

Keywords: approximation algorithm, combination of optimization problems, covering problem, parallel machine scheduling

Mathematics Subject Classification: *68Q25, 68W25, 68W40, 90B35*

1 Introduction

Historically the study of combinatorial optimization was divided into many subfields, e.g. machine scheduling, network flow, network design, knapsack problem and bin packing problem. These subfields were usually studied individually. With the rapid development of science and technology, the decision-makers always need to deal with a system concerned with more than one optimization problem. For example, we want to build a road between two main cities by some construction corporations. The decision-maker needs to choose a path to connect the two cities, that usually is a shortest path problem, and then assign the construction tasks to the corporations, that usually corresponds to a machine scheduling problem. How can he or she choose a feasible path that can be constructed as early as possible? This problem combines the shortest path problem and the machine scheduling problem. Many combination problems can be found in real life, but little research studied the combination of optimization problems in literature.

Wang and Cui first presented the combination problem of parallel machine scheduling and the vertex cover problem in which some of the jobs need to be assigned on m identical parallel machines such that the assigned jobs correspond to a vertex cover for a given

*Corresponding author. Wang's research work has been supported by NSFC No. 11371216 and 11171177, and Bilateral Scientific Cooperation Project between Tsinghua University and KU Leuven.

graph and the objective is to minimize the makespan [30]. They denoted the problem by P_m|vertex cover|C_{max}, and revealed that it may lead to an inefficient solution if we simply solve the two subproblems successively. They proposed an LLR algorithm that incorporates the LPT (Longest Processing Time) rule for parallel machine scheduling and the local ratio method for the vertex cover problem by introducing a replacement policy, and proved that the LLR algorithm is $(3 - \frac{2}{m+1})$-*approximate*. This study connects two fundamental combinatorial optimization problems, and inspires us to consider a wide range of new combinations of combinatorial optimization problems. There are some natural extensions of this work, for instance we can consider the parallel machines to be uniformly related, and the vertex cover problem can be replaced by one of its extensions, for example the prize collecting vertex cover problem or the hitting set problem. The purpose of this paper is to propose a unified combination problem containing all of these extensions.

We say a combinatorial optimization problem is a covering problem if it is a minimization problem with binary variables, and it minimizes a linear objective function under some constraints. The covering problem generalizes many classic combinatorial optimization problems, e.g. the shortest path problem, the vertex cover problem and the hitting set problem. In fact, the combinatorial optimization problem defined by Wolsey coincides with our covering problem [31]. Our combination problem is to assign some of the jobs on m uniformly related parallel machines such that the jobs correspond to a feasible solution of the covering problem and the makespan is minimized. We denote the combination problem by Q_m|COVERING|C_{max}. The combination problem is usually not easier than any of the individual problems, and it needs to investigate the properties of the individual problems and integrate them to develop an efficient algorithm for the combination problem. We first give a short review on scheduling problems. The covering problem and the combination problem will be further discussed after we formally define them in the next section.

Scheduling problems have wide applications in many areas, including computer and manufacturing systems, agriculture, hospital, transport and so on [26]. Scheduling problems are usually specified by the machine environment, the optimally criterion and the job characteristics [5]. In a uniformly related parallel machine scheduling problem, denoted by $Q_m||C_{max}$, each machine has a processing speed and the objective is to minimize the makespan, i.e. the last completion time. If the speeds are the same for all machines, $Q_m||C_{max}$ becomes the identical parallel machine scheduling problem, denoted by $P_m||C_{max}$. Both of $P_m||C_{max}$ and $Q_m||C_{max}$ are NP-hard [11], and they have been extensively studied. In the first paper on the worst-case analysis of an approximation algorithm, Graham showed that the LS (List Scheduling) algorithm, which assigns the first available job on the least load machine, is $(2 - \frac{1}{m})$-*approximate* for $P_m||C_{max}$ [13]. If the jobs are ordered of non-increasing processing times, then the LS algorithm is known as the LPT rule. Graham proved that the LPT rule has a worst case ratio $\frac{4}{3} - \frac{1}{3m}$ for $P_m||C_{max}$ [14]. Morrison applied the LPT rule to $Q_m||C_{max}$ and obtained a $\max\{\frac{\sigma}{2}, 2\}$-*approximation* algorithm in which σ is the ratio of the maximum machine speed to the minimum machine speed [28]. Gonzalez and Ibarra presented a modified LPT rule that assigns the current job to the machine with the smallest completion time, and showed that its worst-case ratio is $2 - \frac{2}{m+1}$ [12]. A constant ratio $\frac{19}{12}$ for $Q_m||C_{max}$ was obtained in [9] and [10]. Recently the result was improved by Kovács [25]. Each of $P_m||C_{max}$ and $Q_m||C_{max}$ has PTAS [17, 18], and FPTAS exists if the number of the machines is fixed [29, 20].

The contributions of this paper include: (1) we formally describe the considered problem; (2) given that there is an algorithm with worst-case ratio r for the covering problem, we present two approximation algorithms for Q_m|COVERING|C_{max} with worst-case ratios $r + \frac{(m-1)\lambda_m}{\sum_{i=1}^{m} \lambda_i}$ and $\frac{r+\sqrt{r^2+4r(m-1)}}{2}$ respectively where λ_i is the speed of machine i; (3) if the machines are identical or the covering problem is specified, e.g. shortest path, price collecting vertex cover or hitting set, some explicit or better results are obtained.

The rest of this paper is organized as follows. We formally define the covering problem and the combination problem in Section 2. In Section 3, we propose and analyze the LA_rR algorithm for $Q_m|\text{COVERING}|C_{max}$, and some improved results are obtained in Section 4. In Section 5, we apply our results to some specific combination problems, and Section 6 concludes our work.

2 Preliminaries

We first give the definition of a covering problem.

Definition 2.1. Given a set of feasibility constraints C, a nonnegative weight vector $w = (w_1, \ldots, w_n)$, a problem is called a covering problem if it minimizes $w \cdot x$ by finding a vector $x \in \{0,1\}^n$ such that x satisfies the constraints in C, in which $\cdot$ denotes the inner product of two vectors.

In the rest of the paper, we always use COVERING(w) to denote a covering problem with certain constraints C in which w denotes the non-negative weight vector. When there is no danger of confusion, we also simply write COVERING(w) as COVERING. The covering problem generalizes many classic combinatorial optimizations. For example, in the vertex cover problem, we are given an undirected graph $G = (V, E)$, in which each vertex i has a non-negative weight w_i, the objective is to find a set of vertices $U \subseteq V$ with minimum total weight such that for each edge (i, j) of E, at least one of i and j belongs to U. Let $x_i = 1$ if the vertex i is chosen and $x_i = 0$ otherwise, and then a set of vertices is a feasible solution if and only if the corresponding variables satisfy $x_i + x_j \geq 1$ for each edge (i, j) of E, that specify the constraints of C. Assume $|V| = n$, and then the vertex cover problem is to minimize $w \cdot x$ by finding a vector $x \in \{0,1\}^n$ such that x satisfies the constraints of C. Similarly, we can verify that the shortest spanning tree problem, the shortest path problem, the Steiner tree problem and the extensions of the vertex cover problem, for example the prize collecting vertex cover problem and the hitting set problem, are all covering problems. Since the related bibliography on covering problems is vast, we will only mention some related results in Section 5 when we apply our general result to some specific combination problems, and the readers are referred to [24] for comprehensive reviews.

Now we can define the combination problem of $Q_m||C_{max}$ and a covering problem COVERING.

Definition 2.2. Given a covering problem COVERING with n variables, each variable x_i of COVERING corresponds to a job J_i with processing time p_i. Let $p = (p_1, \ldots, p_n)$ be the vector of processing times. Define S_x to be a set of jobs such that $J_i \in S_x$ if and only if $x_i = 1$. The $Q_m|\text{COVERING}|C_{max}$ problem is to find a feasible solution x of COVERING and assign the jobs of S_x on m uniformly related parallel machines to minimize the makespan.

Throughout this paper, we generally assume that the covering problem COVERING has a feasible solution since otherwise $Q_m|\text{COVERING}|C_{max}$ will have no feasible solution. In $Q_m|\text{COVERING}|C_{max}$, COVERING provides the feasibility constraints and the scheduling problem $Q_m||C_{max}$ decides the final objective. The weight vectors w and p are not necessarily the same in the definition. It seems that the objective of COVERING does not appear in Definition 2.2, but we define the covering problem to be an optimization problem other than a feasibility problem because of two reasons. First, if we set $w = p$ and $m = 1$, then

$Q_m|\text{COVERING}|C_{max}$ is indeed the covering problem. It implies that the optimization problem COVERING is inherent in the considered problem though the objective of COVERING does not directly act on Definition 2.2. Secondly, and the most important reason is that to design and analyze an efficient algorithm for the combination problem, we need COVERING have the following property, which only applies to an optimization problem.

Assumption 2.1. Assume that there exists a polynomial-time r-*approximation* algorithm A_r for the covering problem COVERING.

We define the covering problem in a general way such that many combinatorial optimization problems are special cases of it. Assumption 2.1 provides some useful information on the covering problem. In the later discussion, we will iteratively adopt a r-*approximation* algorithm A_r for a specific covering problem to obtain an efficient solution for the combination problem obtained by combining the parallel machine scheduling problem and the covering problem.

The $P_m|\text{vertex cover}|C_{max}$ problem studied in [30] is a special case of $Q_m|\text{COVERING}|C_{max}$ in which the parallel machines are identical and COVERING is the vertex cover problem. We know that the vertex cover problem has $(2-\frac{\log\log n}{2\log n})$-*approximation* algorithm based on the local ratio method [3] and a $(2-\Theta(\frac{1}{\sqrt{\log n}}))$-*approximation* algorithm by an SDP approach [22].

$Q_m||C_{max}$ is another subproblem of our combination problem. Let the m uniformly related parallel machines be $\{M_1, M_2 \ldots, M_m\}$ with processing speeds $\{\lambda_1, \lambda_2, \ldots, \lambda_m\}$. If $\lambda_1 = \lambda_2 = \cdots = \lambda_m$, then $Q_m||C_{max}$ is reduced to $P_m||C_{max}$. Without loss of generality, assume that $1 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_m$. Let $J = \{J_1, J_2 \ldots, J_n\}$ be a set of jobs with the vector of processing times $p = (p_1, \ldots, p_n)$. The following LPT rule is a practical and efficient algorithm for $P_m||C_{max}$.

Algorithm 1 The LPT rule for $P_m||C_{max}$

1: sort the jobs in order of non-increasing processing times,
2: whenever a machine becomes idle, assign the first available job to it.

For $Q_m||C_{max}$, the LPT rule is stated as follows.

Algorithm 2 The LPT rule for $Q_m||C_{max}$

1: sort the jobs in order of non-increasing processing times,
2: assign the first available job to the machine on which it will finish first.

The LPT rule will be frequently used as subroutines in our algorithms.

3 Approximation Algorithm for the Combination Problem

For $P_m|\text{vertex cover}|C_{max}$, Wang and Cui developed the LLR algorithm that incorporates the LPT rule for $P_m||C_{max}$ and the local ratio method for the vertex cover problem by a replacement policy [30]. The LLR algorithm first adopts the local ratio method for the vertex cover problem to obtain a feasible solution, and then schedules the corresponding jobs by the LPT rule. While the last completed job occupies one machine, the LLR algorithm considers to replace it by its neighbors and schedules the jobs by the LPT rule iteratively. The work on $P_m|\text{vertex cover}|C_{max}$ inspires the current study, but the main ideas of the

LLR algorithm do not work for $Q_m|\text{COVERING}|C_{max}$ because it uses a specific algorithm and some underlying properties of the vertex cover problem whereas our covering problem is abstract and we only know it has a r-*approximation* algorithm. Notice that different choices of the weight vector w in COVERING do not act on any feasible solution. We can first set $w = p$ and adopt the A_r algorithm to find a r-*approximation* solution x for COVERING, and then schedule the jobs of S_x by the LPT rule. To improve the current solution, we can revise the weight vector w, and obtain another feasible solution by the A_r algorithm, and then run the LPT rule again. We can expect a good solution by iteratively running this procedure if we can establish an appropriate policy to determine when and how to revise the weight vector. Our optimism comes from the fact that we will iteratively run the A_r algorithm for COVERING whereas the LLR algorithm only runs the local ratio method once for the vertex cover problem. We propose the LA_rR algorithm that iteratively runs the A_r algorithm for COVERING and the LPT rule for $Q_m||C_{max}$ by adopting the following revision policy: in a current schedule, if a job, namely J_s, whose processing time p_s is big enough with respect to the current makespan, we will set w_s to a big enough value to evade J_s's appearance again.

Before formally giving the LA_rR algorithm, we first define some symbols and notations. Let $p(S)$ be the total weights of jobs in a job set S. Use $(S'_1, S'_2, \ldots, S'_m)$ to record a current schedule with makespan C'_{max} and use $(S_1, S_2, \ldots, S_m)$ to record a best schedule so far with makespan C_{max} in which S_i is a set of jobs that are assigned on machine i. Let S^*_i be the set of jobs that are assigned on machine i in an optimal solution. Denote by $(S^*_1, S^*_2, \ldots, S^*_m)$ an optimal solution with makespan C^*_{max}, and let x^* be the corresponding feasible solution for COVERING. Define $S = \cup_{i=1}^m S_i$ and $S^* = \cup_{i=1}^m S^*_i$. In our algorithm, we use D to record a set of jobs such that $J_j \in D$ if and only if w_j has been revised. Let $k > 0$ be a constant, and its exact expression will be given later. The algorithm LA_rR can be described as follows.

Algorithm 3 The LA_rR algorithm for $Q_m|\text{COVERING}|C_{max}$

set $w = p$, apply the A_r algorithm for COVERING(w) to obtain a solution x, and construct S_x.
schedule the jobs of S_x by the LPT rule, obtain a schedule $\{S'_1, S'_2, \ldots, S'_m\}$ with makespan C'_{max} and let J_s be the job with the largest processing time in S_x.
let $\{S_1, S_2, \ldots, S_m\} = \{S'_1, S'_2, \ldots, S'_m\}$, $C_{max} = C'_{max}$, $K = r\sum_{i=1}^n |p_i| + 1$ and $D = \emptyset$.
while $p_s \geq \frac{C'_{max}}{k}$ and $w \cdot x < K$ **do**
  $w_s \leftarrow K$, $D \leftarrow D \cup \{J_s\}$.
  apply the A_r algorithm to obtain a solution x for COVERING(w), and construct S_x.
  schedule the jobs of S_x by the LPT rule, obtain $(S'_1, S'_2, \ldots, S'_m)$, C'_{max} and let J_s be the job with the largest processing time in S_x.
  if $C'_{max} < C_{max}$ **then**
    $(S_1, S_2, \ldots, S_m) \leftarrow (S'_1, S'_2, \ldots, S'_m)$, $C_{max} \leftarrow C'_{max}$.
  end if
end while
return $(S_1, S_2, \ldots, S_m)$ and C_{max}.

Let C_{max} be the makespan returned by the LA_rR algorithm. A simple observation is that $C_{max} \leq C'_{max}$ always holds for any current schedule with makespan C'_{max} since the schedule returned is the best one during the LA_rR algorithm.

The feasibility of COVERING does not change if we revise the weight vector w. Since we have assumed that COVERING has a feasible solution, the LA_rR algorithm will return a feasible solution for $Q_m|\text{COVERING}|C_{max}$. In the algorithm, if $w \cdot x \geq K$, then the algorithm stops. It implies that each weight of w can be revised at most once, and hence there are at most n iterations in which the A_r algorithm and the LPT rule are called. The LPT rule can be realized in $O(n \log n + mn)$ time. Assume that the time complexity of the A_r algorithm is T_r, and then the time complexity of the LA_rR algorithm is $O(nT_r + n^2 \log n + mn^2)$.

Before analyzing the performance of the LA_rR algorithm, we prove two useful lemmas.

Lemma 3.1. *If $D \cap S^* \neq \emptyset$, then the LA_rR algorithm is* $k\lambda_m$-approximate *for* $Q_m|\text{COVERING}|C_{max}$.

Proof. Suppose $J_i \in D \cap S^*$, then

$$C^*_{max} \geq \frac{p_i}{\lambda_m}. \tag{3.1}$$

Since $J_i \in D$, there must be a current schedule with makespan C'_{max} such that

$$p_i \geq \frac{C'_{max}}{k}, \tag{3.2}$$

and then J_i is put into D. Noticing that $C_{max} \leq C'_{max}$ for any current schedule with makespan C'_{max}, we have

$$k\lambda_m C^*_{max} \geq C_{max}, \tag{3.3}$$

and the result follows. □

Lemma 3.2. *Let S_x be the set of jobs in the last current schedule, and w be the corresponding weight vector of* COVERING. *If $D \cap S^* = \emptyset$, then $p(S_x) = w \cdot x \leq rp(S^*)$.*

Proof. We obtain x by applying the A_r algorithm for COVERING(w). Assume that $\tilde{x}$ is the optimal solution for COVERING(w), and then $w \cdot x \leq rw \cdot \tilde{x}$. If $D \cap S^* = \emptyset$, we know the weights of jobs in S^* is not changed during the algorithm, and thus x^* satisfies $p(S^*) = p \cdot x^* = w \cdot x^*$. Since we set $K = r\sum_{i=1}^{n}|p_i| + 1$, we have

$$w \cdot x \leq rw \cdot \tilde{x} \leq rp(S^*) < K, \tag{3.4}$$

which implies that the weights of jobs in S_x are not changed, i.e. $w \cdot x = p \cdot x$. Notice that $p(S_x) = p \cdot x$, and the result follows. □

Now we can present our main result.

Theorem 3.3. *Set $k = \frac{r}{\lambda_m} + \frac{m-1}{\sum_{i=1}^{m}\lambda_i}$ in the LA_rR algorithm, and then the LA_rR algorithm is $k\lambda_m$*-approximate *for* $Q_m|\text{COVERING}|C_{max}$.

Proof. We analyze the performance of the LA_rR algorithm under two cases.

Case 1: $D \cap S^* \neq \emptyset$

Lemma 3.1 implies that the LA_rR algorithm is $k\lambda_m$*-approximate.*

Case 2: $D \cap S^* = \emptyset$

In this case, let $(S'_1, S'_2, \ldots, S'_m)$ be the last current schedule with job set S_x and makespan C'_{max}. Since $D \cap S^* = \emptyset$, from Lemma 3.2, we know $w \cdot x = p(S_x) = p \cdot x < K$ must hold. Therefore, we have $p_s < \frac{C'_{max}}{k}$ in the last current schedule since otherwise the "while" iteration will continue. Let J_l be the last completed job in this schedule. Obviously, we have

$$p_l < \frac{C'_{max}}{k}. \tag{3.5}$$

On the other hand, noticing that $C^*_{max} \geq \frac{p(S^*)}{\sum_{i=1}^m \lambda_i}$, by lemma 2, we have

$$C^*_{max} \geq \frac{p(S_x)}{r\sum_{i=1}^m \lambda_i}. \tag{3.6}$$

Suppose J_l is assigned on the hth machine with completion time C'_{max}, the LPT rule implies that $\forall i \neq h$,

$$\frac{p(S'_i) + p_l}{\lambda_i} \geq C'_{max}. \tag{3.7}$$

Therefore, we have

$$p(S_x) = \sum_{i=1}^m p(S'_i) \geq \sum_{i \neq h} (\lambda_i C'_{max} - p_l) + \lambda_h C'_{max} = \sum_{i=1}^m \lambda_i C'_{max} - (m-1)p_l. \tag{3.8}$$

Putting (3.6) and (3.8) together, we have

$$C^*_{max} \geq \frac{\sum_{i=1}^m \lambda_i C'_{max} - (m-1)p_l}{r\sum_{i=1}^m \lambda_i}, \tag{3.9}$$

and thus

$$\frac{C'_{max}}{C^*_{max}} \leq \frac{r\sum_{i=1}^m \lambda_i C'_{max}}{\sum_{i=1}^m \lambda_i C'_{max} - (m-1)p_l}. \tag{3.10}$$

From (3.5) and (3.10), we have

$$\frac{C'_{max}}{C^*_{max}} \leq \frac{rk\sum_{i=1}^m \lambda_i}{k\sum_{i=1}^m \lambda_i - (m-1)}. \tag{3.11}$$

We have indicated that $C_{max} \leq C'_{max}$, and then we can conclude that the LA_rR algorithm has a worst-case ratio $\max\{k\lambda_m, \frac{rk\sum_{i=1}^m \lambda_i}{k\sum_{i=1}^m \lambda_i - (m-1)}\}$. Simple calculation implies that the two terms are all equal to $r + \frac{(m-1)\lambda_m}{\sum_{i=1}^m \lambda_i}$ when $k = \frac{r}{\lambda_m} + \frac{m-1}{\sum_{i=1}^m \lambda_i}$. Therefore, the LA_rR algorithm is $k\lambda_m$-*approximate* for $Q_m|\text{COVERING}|C_{max}$. □

4 Improved Approximation Algorithms

In this section, we propose two improved approximation algorithms for $Q_m|\text{COVERING}|C_{max}$ and $P_m|\text{COVERING}|C_{max}$ respectively. The first approximation algorithm is based on a simple observation that the LA_rR algorithm may perform badly if the speed of the last machine, namely λ_m, is big enough with respect to $\sum_{i=1}^m \lambda_i$. It is possible to obtain an improved solution for this case if we simply assign all jobs of a feasible solution to the last machine. For $P_m|\text{COVERING}|C_{max}$, we can directly adopt the result of Theorem 3.3 by setting $\lambda_1 = \cdots = \lambda_m = 1$, but we make use of the property of identical parallel scheduling to obtain improved result.

4.1 The LLA_rR Algorithm for $Q_m|\text{covering}|C_{max}$

We first present an algorithm, named the LAZY algorithm, that runs in a simple way.

Obviously, the time complexity of the LAZY algorithm is $O(T_r(n)+m+n)$. The analysis of this algorithm is also simple. Actually we have

Algorithm 4 The LAZY algorithm for $Q_m|\text{COVERING}|C_{max}$

1: set $w = p$, apply the A_r algorithm for COVERING(w) to obtain a solution x, and construct S_x.
2: assign all jobs in S_x to the mth machine.
3: **return** $\{0, \ldots, 0, S_x\}$, $C_{max} = p(S_x)/\lambda_m$.

Theorem 4.1. *The LAZY algorithm is* $\frac{r\sum_{i=1}^{m}\lambda_i}{\lambda_m}$-approximate *for* $Q_m|\text{COVERING}|C_{max}$.

Proof. The solution x is obtained by applying the A_r algorithm for COVERING(p), and thus $p(S_x) \le rp(S^*)$. Since $C_{max} = p(S_x)/\lambda_m$ and $C^*_{max} \ge \frac{p(S^*)}{\sum_{i=1}^{m}\lambda_i}$, we have

$$\frac{C_{max}}{C^*_{max}} = \frac{p(S_x)}{\lambda_m C^*_{max}} \le \frac{\sum_{i=1}^{m}\lambda_i p(S_x)}{\lambda_m p(S^*)} \le \frac{r\sum_{i=1}^{m}\lambda_i}{\lambda_m}, \tag{4.1}$$

and then the result follows. □

Although the LAZY algorithm is much simpler than the LA_rR algorithm, it may perform better in case λ_m is big enough. A natural idea is to run the two algorithms consecutively and output the better solution, that leads to the LLA_rR algorithm.

Algorithm 5 The LLA_rR algorithm for $Q_m|\text{COVERING}|C_{max}$

1: apply the LAZY algorithm to get a solution $(S_1, S_2, \ldots, S_m)$ with makespan C_{max}.
2: apply the LA_rR algorithm to get a solution $(S'_1, S'_2, \ldots, S'_m)$ with makespan C'_{max}.
3: **if** $C_{max} \le C'_{max}$ **then**
4:    **return** $(S_1, S_2, \ldots, S_m)$ and C_{max}.
5: **else**
6:    **return** $(S'_1, S'_2, \ldots, S'_m)$ and C'_{max}.
7: **end if**

The time complexity of the LLA_rR algorithm is $O(n(T_r(n) + n\log n + mn))$. Combining the results of Theorems 3.3 and 4.1, we have

Theorem 4.2. *The* LLA_rR *algorithm is* $\min\{\frac{r\sum_{i=1}^{m}\lambda_i}{\lambda_m}, r + \frac{(m-1)\lambda_m}{\sum_{i=1}^{m}\lambda_i}\}$-approximate *for* $Q_m|\text{COVERING}|C_{max}$.

Let $\alpha = \min\{\frac{r\sum_{i=1}^{m}\lambda_i}{\lambda_m}, r + \frac{(m-1)\lambda_m}{\sum_{i=1}^{m}\lambda_i}\}$, and then we have

$$\alpha \le r + \frac{r(m-1)}{\alpha}, \tag{4.2}$$

which implies that

$$\alpha \le \frac{r + \sqrt{r^2 + 4r(m-1)}}{2}. \tag{4.3}$$

This result gives another worst case ratio of the LLA_rR algorithm for $Q_m|\text{COVERING}|C_{max}$, which is independent of the machine speeds.

Corollary 4.3. *The* LLA_rR *algorithm is* $\frac{r+\sqrt{r^2+4r(m-1)}}{2}$-approximate *for* $Q_m|\text{COVERING}|C_{max}$.

4.2 The LA_rR Algorithm for $P_m|$covering$|C_{max}$

For $P_m|\text{COVERING}|C_{max}$, if we adopt the result of Theorem 3.3 by setting $\lambda_1 = \cdots = \lambda_m = 1$ and $k = r + 1 - \frac{1}{m}$, we know the worst case ratio of the LA_rR algorithm for $P_m|\text{COVERING}|C_{max}$ will be $r + 1 - \frac{1}{m}$. In this subsection, we reduce the approximation factor by investigating the property of identical parallel machine scheduling. We also adopt the LA_rR algorithm for $P_m|\text{COVERING}|C_{max}$, but the choice of k is different. We first give a lemma concerned with the LPT rule for identical parallel machine scheduling. Let $J = \{J_1, J_2 \dots, J_n\}$ be a set of jobs with the vector of processing times $p = (p_1, \dots, p_n)$. Without loss of generality, assume that $p_1 \geq p_2 \geq \cdots \geq p_n$. Let C_{max} be the makespan of the schedule obtained by applying the LPT rule to the jobs of J on m identical parallel machines, and J_l be the last completed job.

Lemma 4.4. *For $P_m||C_{max}$, let C_{max} be the makespan returned by the LPT rule. Given any positive integer k, if $p_1 < \frac{C_{max}}{k}$, then we have*

$$p_l \leq \frac{C_{max}}{k+1}. \tag{4.4}$$

Proof. Let M_i be the machine on which J_l is processed, i.e. the completion time of M_i is C_{max}. Since $p_1 < \frac{C_{max}}{k}$ and $p_1 \geq p_2 \geq \cdots \geq p_n$, at least $k+1$ jobs are processed on M_i. Since J_l is the last completed job on M_i, the LPT rule implies that the processing time of J_l must be the smallest among all jobs processed on M_i. Therefore, we have $p_l \leq \frac{C_{max}}{k+1}$. □

Using this property of the LPT rule, we can obtain an improved result for $P_m|\text{COVERING}|C_{max}$.

Theorem 4.5. *Given any positive integer k in the LA_rR algorithm, the LA_rR algorithm is* $\max\{k, \frac{r(k+1)m}{km+1}\}$*-approximate for $P_m|\text{COVERING}|C_{max}$.*

Proof. Notice that $P_m|\text{COVERING}|C_{max}$ is actually a special case of $Q_m|\text{COVERING}|C_{max}$, and then all results for $Q_m|\text{COVERING}|C_{max}$ hold when we apply the LA_rR algorithm for $P_m|\text{COVERING}|C_{max}$.

Case 1: $D \cap S^* \neq \emptyset$

In this case, we have concluded that LA_rR algorithm is $k\lambda_m$*-approximate* in Theorem 3.3. For our case, the approximation factor is k because of $\lambda_m = 1$.

Case 2: $D \cap S^* = \emptyset$

Let $(S'_1, S'_2, \dots, S'_m)$ be the last current schedule with job set S_x and makespan C'_{max}. Let J_s be the job with the largest processing time in S_x and J_l be the last completed job. The argument in the proof of Theorem 3.3 implies that $p_s < \frac{C'_{max}}{k}$ if $D \cap S^* = \emptyset$. By lemma 4.4, we have

$$p_l \leq \frac{C'_{max}}{k+1}. \tag{4.5}$$

Notice that (3.10) also holds for this case. Substituting $1 = \lambda_1 = \lambda_2 = \cdots = \lambda_m$ into (3.10), we have

$$\frac{C'_{max}}{C^*_{max}} \leq \frac{rmC'_{max}}{mC'_{max} - (m-1)p_l}. \tag{4.6}$$

Combining (4.5) and (4.6), we have

$$\frac{C'_{max}}{C^*_{max}} \leq \frac{rm(k+1)}{mk+1}. \tag{4.7}$$

Since $C_{max} \leq C'_{max}$, we conclude that the LA_rR algorithm is $\max\{k, \frac{r(k+1)m}{km+1}\}$-*approximate* for $P_m|$COVERING$|C_{max}$. □

Notice that Theorem 4.2 holds for any positive integer k, and then we can calculate the best k such that $\max\{k, \frac{r(k+1)m}{km+1}\}$ is minimized. It may lead to a better result than $r+1-\frac{1}{m}$ which is obtained directly by setting $1 = \lambda_1 = \lambda_2 = \cdots = \lambda_m$ in Theorem 3.3. In fact, we have the next theorem.

Theorem 4.6. *There exists a positive integer k such that* $\max\{k, \frac{r(k+1)m}{km+1}\} \leq r+1-\frac{1}{m}$.

Proof. Set $k = \lfloor r+1-\frac{1}{m} \rfloor$ where $\lfloor x \rfloor$ denotes the largest integer not greater than x. Since r is the worst case ratio of the A_r algorithm for a minimization problem, we know $r \geq 1$ and thus $k \geq 1$. Obviously, we have $k \leq r+1-\frac{1}{m}$. Let $h(x) = \frac{r(x+1)m}{xm+1}$, and it is easy to verify that $h(x)$ is a decreasing function for $x \geq 1$. Notice that $k \geq r - \frac{1}{m}$, and we have

$$h(k) \leq h(r-\frac{1}{m}) = \frac{r(r-\frac{1}{m}+1)m}{m(r-\frac{1}{m})+1} = r+1-\frac{1}{m}. \tag{4.8}$$

Therefore, the positive integer k satisfies $\max\{k, \frac{r(k+1)m}{km+1}\} \leq r+1-\frac{1}{m}$.

Now we give the exact expression of positive integer k that minimizes $\max\{k, \frac{r(k+1)m}{km+1}\}$. Let $f(x) = x$, which is a increasing function. Remember that $h(x)$ is a decreasing function for $x \geq 1$. The equation $f(x) = h(x)$ has a unique solution $x^* = \frac{rm-1+\sqrt{4rm^2+(rm-1)^2}}{2m}$ for $x \geq 1$. We conclude that if $f(\lfloor x^* \rfloor + 1) \geq h(\lfloor x^* \rfloor)$, then the optimal k is $\lfloor x^* \rfloor$, and otherwise the optimal k is $\lfloor x^* \rfloor + 1$. □

5 Some Applications of Our Work

In this section, we apply the theoretical results to some specific combination problems. We consider the combinations of identical parallel machine scheduling and three covering problems, namely the shortest path problem, the prize collecting vertex cover problem and the hitting set problem. Denote the three combination problems by $P_m|$shortest path$|C_{max}$, $P_m|$PCVC$|C_{max}$ and $P_m|$hitting set$|C_{max}$ respectively.

5.1 $P_m|$shortest path$|C_{max}$

In a shortest path problem, we are given an undirected graph $G = (V, E)$ and a non-negative weight vector w of edges, and the objective is to find a path between two fixed vertices s and t such that the sum of the weights of its constituent edges is minimized. The shortest path problem is indisputably one of the fundamental problems in computer science. It is well known that Dijkstra algorithm solves the shortest path problem with nonnegative edge weights in polynomial time [7]. The readers are referred to the excellent textbook [1] for more details.

In the LA_rR algorithm, let the A_r algorithm be Dijkstra algorithm in which $r = 1$, and set $k = 1$ by Theorem 4.6. Theorem 4.5 implies that the LA_rR algorithm is $\frac{2m}{m+1}$-*approximate* for $P_m|$shortest path$|C_{max}$. The following instance shows the bound is tight.

Consider an undirected graph $G = (V, E)$ with $2m+1$ nodes and $2m+1$ edges such that $V = \{s, u_1, u_2, \ldots, u_m, t, v_1, v_2, \ldots, v_{m-1}\}$ and $E = E_1 \cup E_2$ in which $E_1 = \{(s, u_1), (u_1, u_2) \ldots, (u_{m-1}, u_m), (u_m, t)\}$ and $E_2 = \{(s, v_1), (v_1, v_2), \ldots, (v_{m-2}, v_{m-1}),$ $(v_{m-1}, t)\}$. Each edge (i, j) corresponds to a job J_{ij} whose processing time is m if $(i, j) \in E_1$

and is $m+1+\epsilon$ if $(i,j) \in E_2$ where $\epsilon > 0$ is a constant. The LA_rR algorithm first sets the weights of edges equal to the processing times of corresponding jobs, and then finds the $s-t$ shortest path containing all edges of E_1 by Dijkstra algorithm, and then assigns the corresponding jobs on m machines by the LPT rule to obtain a schedule with $C'_{max} = 2m$. Since the largest processing time of these jobs is m and $k = 1$, we have $m < \frac{C'_{max}}{k}$, and thus the LA_rR algorithm will output the current schedule with $C_{max} = 2m$. Obviously the optimal makespan of this instance is $m+1+\epsilon$. Thus $\frac{C_{max}}{C^*_{max}} \to \frac{2m}{m+1}$ when $\epsilon \to 0$ for the instance.

5.2 $P_m|\mathbf{PCVC}|C_{max}$

The prize collecting vertex cover problem (PCVC for short) is an extension of the classic vertex cover problem (VC for short). In this problem, we are given an undirected graph $G = (V, E)$ just like the classic vertex cover problem, but now, not only each vertex but also each edge has a nonnegative weight, our target is to find a set U consisting of some edges and some vertices in $V \cup E$, such that for each edge in the graph, either the edge is in U, or at least one of its two endpoints belong to U, and we aim to minimize the total weight of the elements in U.

Hochbaum first introduced PCVC, and obtained 2-*approximation* algorithm [16]. Bar-Yehuda and Rawitz [4] obtained the same approximation factor by the local ratio method, which is remarkably simple and elegant. Since VC is a special case of PCVC, and hence the following inapproximability results for VC also holds for PCVC. Dinur and Safra proved that the vertex cover problem can not be approximated within a factor of 1.36 unless $P = NP$ [8]. Based on the unique games conjecture (UGC), Khot and Vygen showed that VC can not be approximated within any constant factor better than 2 [23].

In the LA_rR algorithm, let the A_r algorithm be the local ratio method of Bar-Yehuda and Rawitz in which $r = 2$, and set $k = 2$ by Theorem 4.6. Theorem 4.5 implies that the LA_rR algorithm is $\frac{6m}{2m+1}$-*approximate* for $P_m|\text{PCVC}|C_{max}$. Before giving an instance to show the bound is tight, we describe the local ratio method of Bar-Yehuda and Rawitz.

Algorithm 6 The local ratio method for prize collecting vertex cover [4]

```
while there exists an edge e = (u, v) such that min{w(u), w(v), w(e)} > 0 do
    ε = min{w(u), w(v), w(e)},
    w(u) ← w(u) − ε, w(v) ← w(v) − ε, w(e) ← w(e) − ε.
end while
return {v|w(v) = 0} ∪ {e = (u, v)|w(u) > 0, w(v) > 0}.
```

Consider a complete undirected graph with $2m+1$ vertices. Each vertex i corresponds to a job with processing time $2m$, and each edge (i,j) corresponds to a job with processing time 1. The LA_rR algorithm first sets the weights of vertices and edges equal to the processing times of corresponding jobs, and then finds a feasible cover containing all vertices by the local ratio method, and then assigns the corresponding jobs on m machines by the LPT rule to obtain a schedule with $C'_{max} = 6m$. Since the largest processing time of these jobs is $2m$, we have $2m < \frac{C'_{max}}{2}$, and thus the LA_rR algorithm will output the current schedule with $C_{max} = 6m$. The optimal solution is to select all of the $m(2m+1)$ edges that cover the graph and then assign the corresponding $2m+1$ jobs of them on each machine. The optimal makespan is $2m+1$, and thus $\frac{C_{max}}{C^*_{max}} = \frac{6m}{2m+1}$ for the instance.

Notice that VC is a special case of PCVC, and thus we conclude that the LA_rR algorithm is also $\frac{6m}{2m+1}$-*approximate* for P_m|vertex cover|C_{max}. For the same problem, Wang and Cui developed an LLR algorithm which is $(3-\frac{2}{m+1})$-*approximate* [30], and a recent work reduced the approximation factor to $\frac{5}{2}-\frac{1}{2m}$ for $m=2,3$ and to $3-\frac{3}{m+1}$ for $m\geq 4$ [19]. It is not strange that the result obtained in this paper is not better than that of [30, 19] since our result is obtained under a unified framework, whereas the other approaches are dedicated to a specific problem by investigating the underlying properties of the vertex covering problem, parallel machine scheduling and their combination problem.

5.3 P_m|**hitting set**|C_{max}

The hitting set problem (HS for short) is another extension of VC. In HS, we are given a collection of nonempty sets $\{S_1, S_2, \ldots, S_l\}$ and a nonnegative weight vector w of the sets' elements $S=\bigcup_{i=1}^{l} S_i$. A set $U\subseteq S$ is said to hit a given set S_i if $U\cap S_i\neq\emptyset$. Our target is to find a minimum-cost subset of S that hits all the sets S_i, where $i=1,2,\ldots,l$.

Johnson [21] and Lovasz [27] first studied the unweighted case of HS, and presented a greedy algorithm with approximation factor d_{max}, where d_{max} is the maximum degree of an element. Chvatal [6] proved that this result holds for the general case. Based on linear programming, Hochbaum proposed two f-*approximation* algorithms, where $f=max_{1\leq i\leq l}|S_i|$ [15]. Bar-Yehuda and Even developed the following linear time f-*approximation* algorithm by the local ratio method.

Algorithm 7 The local ratio method for hitting set [2]

```
while there exists a set S_i such that min{w(x)|x ∈ S_i} > 0 do
    ε = min{w(x)|x ∈ S_i},
    for each x ∈ S_i, w(x) = w(x) − ε.
end while
return {x|w(x) = 0}.
```

In the LA_rR algorithm, let the A_r algorithm be the local ratio method of Bar-Yehuda and Even in which $r=f$, and set $k=f$ by Theorem 4.6. Theorem 4.5 implies that the LA_rR algorithm is $\frac{f(f+1)m}{fm+1}$-*approximate* for P_m|hitting set|C_{max}. Different from the previous two problems, this bound is not tight. In fact, we can reduce the approximation factor to $f+1-\frac{f}{m(f-1)+1}$ by a careful analysis, but we omit its proof since it deviates from our aim of demonstrating the theoretical results obtained in the last section. Based on UGC, Khot and Vygen showed HS can not be approximated with any constant less than f [23], and thus f is also a threshold for P_m|hitting set|C_{max}. Though the bound $\frac{f(f+1)m}{fm+1}$ is not tight, it is close to the threshold since $f\leq\frac{f(f+1)m}{fm+1}<f+1$.

6 Conclusions

In this paper, we have studied the combination of uniformly related parallel machine scheduling and a generalized covering problem under a unified framework. In the combination problem, the covering problem provides the feasibility constraints for the scheduling problem whereas the objective functions of the covering problem are decided by the scheduling problem in our algorithms. We propose an approximation algorithm that incorporates the LPT rule for uniformly related parallel machine scheduling and the approximation algorithm for the covering problem by introducing a weight revision policy that connects the two problems. We also improve our results for some special cases. We present some specific combination problems of identical parallel machines scheduling and classic combinatorial optimization problems to demonstrate our approach. Many valuable problems remain. For instance, can we obtain improved results for the generalized or specific combination problems? How about the combination problem of the bin packing problem and a covering problem or the combination of other classic combinatorial optimization problems? All these questions deserve further investigation.

Acknowledgments

We would like to thank the anonymous referees for their helpful comments and suggestions.

References

[1] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, 1993.

[2] R. Bar-Yehuda and S. Even, A linear time approximation algorithm for the weighted vertex cover problem, *J. Algorithms* 2 (1981) 198–203.

[3] R. Bar-Yehuda and S. Even, A local-ratio theorem for approximating the weighted vertex cover problem, *Ann. of Discrete Math.* 25 (1985) 27–45.

[4] R.Bar-Yehuda and D. Rawitz, On the equivalence between the primal-dual schema and the local-ratio technique, *In 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, 2001, pp. 24–35.

[5] B. Chen, C. Potts, and G. Woeiginger, A review of machine scheduling: Complexity, algorithm and approximability, *In Handbook of Combinatorial Optimization*, D-Z. Du, P. Pardalos (eds.), Kluwer Academic Publishers, 1998, pp. 21–169.

[6] V. Chvatal, A greedy heuristic for the set-covering problem, *Math. Oper. Res.* 4 (1979) 233–235.

[7] E. Dijkstra, A note on two problems in connexion with graphs, *Numer. Math.* 1 (1959) 269–271.

[8] I. Dinur and S. Safra, On the hardness of approximating minimum vertex-cover, *Ann. of Math.* 162 (2005) 439–485.

[9] G. Dobson, Scheduling independent tasks on uniform processors, *SIAM J. Comput.* 13 (1984) 705–716.

[10] D. Friesen, Tighter bounds for LPT scheduling on uniform processors, *SIAM J. Comput.* 16 (1987) 554–660.

[11] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, WH Freeman, 1979.

[12] T. Gonzalez and O. Ibarra, Bounds for LPT schedules on uniform processors, *SIAM J. Comput.* 6 (1977) 155–166.

[13] R. Graham, Bounds for certain multiprocessing anomalies, *Bell System Technical Journal* 1 (1966) 1563–1581.

[14] R. Graham, Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.* 17 (1969) 416–426.

[15] D. Hochbaum, Approximation algorithms for the set covering and vertex cover problems, *SIAM J. Comput.* 11 (1982) 555–556.

[16] D. Hochbaum, Solving integer programs over monotone inequalities in three variables: A framework of half integrality and good approximation, *European J. Oper. Res.* 140 (2002) 291–321.

[17] D. Hochbaum and D. Shmoys, Using dual approximation algorithms for scheduling problems: Theoretical and practical results, *J. ACM* 34 (1987) 144–162.

[18] D. Hochbaum and D. Shmoys, A polynomial approximation scheme for machine scheduling on uniform processors: Using the dual approximation approach, *SIAM J. Comput.* 17 (1988) 539–551.

[19] W. Hong and Z. Wang, Improved approximation algorithm for the combination of parallel machine scheduling and vertex cover, Working Paper, Tsinghua University (2012).

[20] E. Horowitz and S. Sahni, Exact and approximate algorithms for scheduling nonidentical processors, *J. ACM* 23 (1976) 317–327.

[21] D. Johnson, Approximation algorithms for combinatorial problems, *J. Comput. System Sci.* 9 (1974) 256–278.

[22] G. Karakostas, A better approximation ratio for the vertex cover problem, *ACM Trans. Algorithms* 5 (2009) 41:1–41:8.

[23] S. Khot and O. Regev, Vertex cover might be hard to approximate to within $2-\epsilon$, *J. Comput. System Sci.* 74 (2008) 335–349.

[24] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms (Fifth Edition)*, Springer, 2012.

[25] A. Kovács, New Approximation Bounds for LPT Scheduling, *Algorithmica* 57 (2010) 413–433.

[26] J. Leung, *Handbook of Scheduling*, CRC Press, 2004.

[27] L. Lovasz, On the ratio of optimal integral and fractional covers, *Discrete Math.* 13 (1975) 383–390.

[28] J. Morrison, A note on LPT scheduling, *Oper. Res. Lett.* 7 (1988) 77–79.

[29] S. Sahni, Algorithms for scheduling independent tasks, *J. ACM* 23 (1976) 116–127.

[30] Z. Wang and Z. Cui, Combination of parallel machine scheduling and vertex cover, *Theoret. Comput. Sci.* 460 (2012) 10–15.

[31] L.A. Wolsey, *Integer Programming*, Wisly, 1998.

Manuscript received 8 November 2013
revised 18 March 2014
accepted for publication 25 April 2014

Zhenbo Wang
Department of Mathematical Sciences
Tsinghua University, Beijing, China
E-mail address: zwang@math.tsinghua.edu.cn

Wenyi Hong
Department of Mathematical Sciences
Tsinghua University, Beijing, China
E-mail address: hongwy10@mails.tsinghua.edu.cn

Dong He
Department of Mathematical Sciences
Tsinghua University, Beijing, China
E-mail address: hedongtg@163.com