Efficient Minimization of Sum and Differential Costs on Machines with Job Placement Constraints

Jaya Prakash Champati and Ben Liang
Department of Electrical and Computer Engineering, University of Toronto
{champati,liang}@comm.utoronto.ca

***Abstract*—We revisit the problem of assigning n jobs to m machines/servers. We study this problem under more general settings, which capture important aspects of applications that arise in networking and information systems. In particular, we consider jobs that have placement constraints and machines that are heterogeneous. The cost incurred at a machine is given by any general convex function on the number of jobs assigned to it. We aim to minimize the sum cost and the maximum differential cost. Through a network-flow equivalence transformation, we observe how these two objectives are fundamentally related, showing that sum-cost minimization implies maximum-differential-cost minimization. We propose an efficient algorithm termed Maximum Edge-Cost Cycle Cancelling (MEC^3) to solve the sum-cost minimization problem with $O(n^2m^2)$ time complexity. Furthermore, for applications where only the maximum differential cost is of concern, we further improve the efficiency of MEC^3 by proposing an early stop condition. We implement MEC^3 and two other algorithms from the literature. Using benchmark input instances, we show that MEC^3 has substantially lower run time than the other algorithms.**

I. Introduction

We study the problem of assigning n jobs to m machines. For example, a machine may represent a server, sensor, or storage unit, while a job may represent a user, sensor, or read request. The machines are heterogeneous in the sense that each machine is associated with a different value of some attribute, e.g., processing speed, energy, or bandwidth. The jobs are identical and have placement constraints, i.e., a job can only be assigned to a subset of the machines. The cost incurred at a machine is determined by the load, i.e., the number of jobs assigned, on the machine.

Although the above problem setting appears simple, it arises in a wide range of applications in networking and information systems. For example, in [1] the problem was studied in the context of retrieving data blocks from disks in Video on Demand (VoD) systems. The authors of [2] and [3] considered the problem in peer-to-peer systems. The problem also arose in workload balancing among packet queues in [4], and sub-carrier allocation to users in OFDMA systems [5]. More recently, the problem was also studied for data aggregation in wireless sensor networks [6]. Furthermore, the problem setting is potentially useful for cloud storage systems that store data using replication [7], [8].

The problem has received much theoretical treatment in the literature, under different objectives [9]–[13]. In several works, the makespan minimization problem was considered, which is equivalent to assigning a cost to each machine that is linear in its load and minimizing the maximum cost [9]–[11]. Other works considered minimizing the average delay per job or sum completion time [11], [12]. In this case the cost assigned to a machine is given by a quadratic function. Minimizing the L_p-norm of the loads on the machines was considered in [13]. Since these problem instances are known to be solvable, major research effort has gone into designing algorithms with lower *worst-case* time complexity. However, we note that very few of these works study the actual run time performance of their proposed algorithms.

In contrast to prior works, the authors of [14] studied the problem under a general sum-cost objective, where the cost incurred at any machine is given by a convex function. They proved an interesting result, that an optimal solution for sum-cost minimization is insensitive to the underlying convex cost function. However, we note that in their model, all machines are identical, i.e., the same convex cost function is used for all machines. As a consequence, equal loads on different machines incur equal costs. Since in many practical applications the machines are heterogeneous, the applicability of [14] is limited. For example, quadratic cost functions with machine specific parameters were considered for data aggregation in wireless sensor networks in [6].

This motivates us to study the job assignment problem under more general setting where each machine has its own convex cost function. Under this generalization, we first aim at minimizing the sum cost. Then, we consider the objective of minimizing the maximum differential cost, which we will show to be fundamentally related to sum-cost minimization. We note that the maximum-differential-cost minimization problem is interesting and important in its own right. In fact the widely studied makespan minimization problem is a special case of this general problem.

We study the sum-cost minimization problem by extending the network-flow equivalence framework proposed in [14]. From this equivalence, we obtain the first main result in this paper: *sum-cost minimization implies maximum-differential-cost minimization*. The significance of this result is that the network-flow equivalence, the optimality criterion, and any algorithm proposed for minimizing the sum cost can be used for minimizing the maximum differential cost, under any general convex cost functions of the machines. We note that this result has been proved previously only for specific convex

This work has been supported in part by grants from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

cost functions [6], [14], e.g., it is shown that minimizing the sum completion time (a sum-cost objective) also minimizes the makespan (a maximum-differential-cost objective).

We then propose an algorithm termed Maximum Edge-Cost Cycle Cancelling (MEC^3) to solve the sum-cost minimization problem. We give an efficient implementation of MEC^3 and study its run time performance under the objectives of sum completion time and makespan. For comparison purposes we also implement two other algorithms, which have better worst-case time complexity than MEC^3 but whose run time performance has not been studied before. Using benchmark input instances, we show that MEC^3 is significantly faster than both algorithms. We summarize our main contributions below:

- We study the problem of assigning n jobs to m machines with the objective of minimizing the sum cost, $\sum_i g_i(k)$, where $g_i(k)$ is a general convex cost function associated with the ith machine, and k is the number of jobs assigned to the machine. We also study the problem of minimizing the maximum differential cost given by $\max_i\{g_i(k) - g_i(k-1)\}$. Using a network-flow transformation framework, we show that the solution to the problem of sum-cost minimization also solves the problem of maximum-differential-cost minimization (but not the converse).
- We propose the MEC^3 algorithm to solve the problem of sum-cost minimization, thereby solving the problem of maximum-differential-cost minimization as well. For applications that require only to minimize the maximum differential cost, we further propose an *early stop* condition in MEC^3 that significantly improves its run time in practice. Furthermore, we show that MEC^3 has $O(n^2m^2)$ worst-case time complexity.
- Noting that the CANCELALL algorithm proposed in [15] can be directly extended and used to solve the problem of sum-cost minimization, we compare its run time performance with MEC^3 using benchmark input instances from [16]. Even though CANCELALL has $O(n^{1.5}m\log n)$ time complexity, its average run time is longer by two orders of magnitude compared with that of MEC^3.
- Even though MEC^3 is designed to solve general problems with arbitrary convex machine cost functions, for makespan minimization, which is an important special case of maximum-differential-cost minimization, we show that MEC^3 has much shorter run time than the well-known Maximum-Flow based Makespan Minimization (MFMM) algorithm [9], [11], [17].

The rest of the paper is organized as follows. In Section II we formally define the sum-cost and maximum-differential-cost problems, and discuss various cost functions. In Section III we present the network flow equivalence for sum-cost minimization and derive optimality conditions for both problems. We propose the MEC^3 algorithm, prove its optimality, and derive its time complexity in Section IV. Experimental results are presented in Section V. We discuss the relation between our work and prior works in Section VI, and we conclude in Section VII.

II. Problem Formulation and Applications

In this section we formulate the sum-cost and maximum-differential-cost minimization problems. We discuss special cost functions that have been considered in the literature.

A. Sum Cost and Differential Cost Problems

Consider n jobs and m machines. Let U represent the set of jobs and V represent the set of machines. Let E represent the edge set, where $(u_j, v_i) \in E$ if job $u_j \in U$ can be assigned to machine $v_i \in V$. Thus, E captures the *placement constraints* of the jobs. The system can be represented by the bipartite graph $G = (U \cup V, E)$. We denote the degree of machine v_i by $\deg(v_i)$, which is equal to the maximum number of jobs that can be assigned to machine v_i. In this work we use i to index machine vertices in V and j to index job vertices in U.

A *semi-matching* M is a subset of E such that each job vertex u_j is exactly incident with one edge in M. In other words, a semi-matching assigns each job to only one machine, while respecting the placement constraints specified by E. The *load* on machine v_i under M, denoted by $\deg_M(v_i)$, is the number of edges incident on vertex v_i, i.e., the number of jobs assigned to v_i. Let $g_i(\deg_M(v_i))$ denote the cost incurred at machine v_i, where $g_i : \mathbb{Z}^+ \to \mathbb{R}$ is a convex cost function. In Section II-B, we will discuss different cost functions that may be used in some practical applications. Let $\mathbf{g} = \{g_i(.), \forall i\}$ denote the vector of machine cost functions. The sum cost incurred by a semi-matching M is given by $\text{cost}_\mathbf{g}(M) \triangleq \sum_{i=1}^m g_i(\deg_M(v_i))$. Given $(G, \mathbf{g})$, we are interested in the following *sum-cost* minimization problem $\mathcal{P}_\mathbf{g}^{\text{sum}}$:

$$\underset{M}{\text{minimize}} \quad \text{cost}_\mathbf{g}(M).$$

We also consider the following min-max problem that is closely related to $\mathcal{P}_\mathbf{g}^{\text{sum}}$, and arises in several practical applications. Given a semi-matching M, we refer to $g_i(\deg_M(v_i)) - g_i(\deg_M(v_i)-1)$ as the *differential cost* of vertex v_i, we define the *maximum differential cost* as

$$\Gamma_\mathbf{g}(M) \triangleq \max_{v_i \in V} \{g_i(\deg_M(v_i)) - g_i(\deg_M(v_i) - 1)\}, \quad (1)$$

and we call any vertex a *maximum-differential-cost vertex* in M if it is a maximizer of (1). Given $(G, \mathbf{g})$, we are interested in the following maximum-differential-cost minimization problem $\mathcal{P}_\mathbf{g}^{\max}$:

$$\underset{M}{\text{minimize}} \quad \Gamma_\mathbf{g}(M).$$

An important result in this paper is that an optimal semi-matching for $\mathcal{P}_\mathbf{g}^{\text{sum}}$ is also optimal for $\mathcal{P}_\mathbf{g}^{\max}$.

Remark 1: Our model can accommodate the case where initial loads on the machines are non-zero, i.e., there are jobs already assigned to machines. In this case we solve the problem by simply considering the jobs that are already assigned as part of the input job set, and restrict the placement of each of those jobs to the machine to which it is already assigned.

B. Example Cost Functions

The following example convex cost functions $g_i(.)$ are widely studied in different application scenarios [1], [3], [6], [9], [11], [13]–[15].

Sum completion time and makespan: In this case, we consider jobs that have identical processing requirements and the *uniform machine* model, i.e., the processing time of a job on a machine is inversely proportional to the speed of the machine. Let s_i denote the processing time of any job on a machine v_i. Consider the following quadratic cost function:

$$g_i(k) = \frac{s_i k(k+1)}{2}, \ \forall \ v_i. \tag{2}$$

In this case the sum cost under M is given by

$$\text{cost}_{\mathbf{g}}(M) = \sum_{i=1}^{m} \frac{s_i}{2} \text{deg}_M(v_i)(\text{deg}_M(v_i)+1), \tag{3}$$

which represents the *sum completion time*, i.e, $\frac{\text{cost}_{\mathbf{g}}(M)}{n}$ gives the average completion time of any job. Furthermore, the maximum differential cost is given by

$$\Gamma_{\mathbf{g}}(M) = \max_{v_i \in V} \ s_i \text{deg}_M(v_i). \tag{4}$$

Note that $s_i \text{deg}_M(v_i)$ quantifies the time at which machine v_i completes the processing of jobs assigned to it. Therefore, in this case $\Gamma_{\mathbf{g}}(M)$ represents the *makespan*.

L_p-norm: One may also consider $g_i(\text{deg}_M(v_i)) = (s_i \text{deg}_M(v_i))^p$, for all v_i. In this case minimizing the sum-cost is equivalent to minimizing the L_p-norm on the completion times of the machines [13].

III. Network-Flow Equivalence and Optimality Conditions for $\mathcal{P}_{\mathbf{g}}^{\text{SUM}}$ and $\mathcal{P}_{\mathbf{g}}^{\text{MAX}}$

In this section we first present the transformation of $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$ to a min-cost flow problem. We then give an optimality criterion using Cost Reducing Paths (CRPs). We finally show that an optimal semi-matching for $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$ is also optimal for $\mathcal{P}_{\mathbf{g}}^{\text{max}}$.

A. Transformation to Min-cost Flow Problem in 5-Partite Graph

In this section we focus on the transformation of $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$ to a min-cost network flow problem. We will also discuss $\mathcal{P}_{\mathbf{g}}^{\text{max}}$ in relation to the the network flow graph. We use a network flow transformation framework similar to the one proposed in [14]. However, in [14] the problem was studied for identical machines, i.e., $s_i = 1$ and (2) becomes $g_i(k) = k(k+1)/2$, for all v_i. As we will see later, their framework requires a lower number of edges with non-zero costs, and edge costs are positive integers independent of the machine cost function. In contrast, to accommodate the heterogeneous machine cost functions $g_i(.)$, in our transformation we require a larger number of edges with non-zero costs, and we also need to design edge costs specific to the machine cost functions.

Given the bipartite graph $G = (U \cup V, E)$, we construct a flow network $\mathcal{N}$ as shown in Figure 1. Direct all the edges E from U to V. For each vertex v_i, add vertices $\{c_{i1}, c_{i2}, \ldots, c_{i\text{deg}(v_i)}\}$ and place directed edges from vertex v_i to these new vertices. Let $C = \cup_{i=1}^{m} \{c_{i1}, c_{i2}, \ldots, c_{i\text{deg}(v_i)}\}$. They are called *cost centers* in this work. Add a source and place directed edges from the source to all vertices from U. Also, add a sink and place directed edges from all vertices in C to the sink. The resultant graph is a 5-partite directed graph $(\{\text{source}\} \cup U \cup V \cup C \cup \{\text{sink}\}, E')$, where E' is the edge set of $\mathcal{N}$.

The source vertex is assigned a supply of n units and the sink vertex is assigned a demand of n units, while all other nodes have a demand of zero. We set the capacity of all edges to one. We also set all edges to have cost zero except for the edges from V to C. The cost of edge $(v_i, c_{ik}) \in E'$ is set to

$$\beta_i(k) = g_i(k) - g_i(k-1), \quad 1 \leq k \leq \text{deg}(v_i). \tag{5}$$

Note that the number of edges with non-zero costs are equal to the number of edges in G, i.e, $|C| = |E|$.

We note that the edge costs $\beta_i(k)$ are non-decreasing, i.e.,

$$\beta_i(k+1) - \beta_i(k) \geq 0, \ \forall \ 1 \leq k < \text{deg}(v_i). \tag{6}$$

The above property holds because $g_i(.)$ is convex. Furthermore, if $g_i(k) = k(k+1)/2$, for all v_i, then the cost $\beta_i(k) = k$, is fixed for all i. It can be shown that in this case the cost centers can be merged, so that only $\max_{v_i} \text{deg}(v_i)$ number of cost centers are needed, and the transformation here is reduced to the transformation given in [14].

Next, we present the equivalence between $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$ and a min-cost flow problem in $\mathcal{N}$. Note that any feasible flow in $\mathcal{N}$ carries n units of flow from the source to the sink. The total capacity of edges from the source is n. Therefore, under a feasible flow, each edge from the source should carry one unit of flow to vertices in U. Since each job vertex u_j has demand zero, the unit of flow entering a job vertex u_j must be forwarded to some neighbor machine vertex from V. Therefore, we can obtain a semi-matching by simply enumerating all those edges between U and V that have a unit flow under the feasible flow. This result is stated in Lemma 1 and is a direct consequence of Lemma 2.1 [14].

Lemma 1. *If f is a feasible flow in $\mathcal{N}$, then f determines a unique semi-matching M in G.*

Note that the converse of the above lemma is not true. Given a semi-matching, there may be more than one feasible flows that correspond to the semi-matching. These different feasible flows will have exactly the same flow values on the edges between the source and V but differ in the flow values on the edges between V and C. In the following lemma, we relate the total cost of a feasible flow in $\mathcal{N}$ and the sum-cost objective achieved by the semi-matching determined by the feasible flow.

Lemma 2. *The total cost of any feasible flow f that determines a semi-matching M is lower bounded by $\text{cost}_{\mathbf{g}}(M) - \sum_{i=1}^{M} g_i(0)$. Further, there exists a feasible flow f^M that achieves this lower bound.*

Proof. The proof is given in Appendix-A of [18]. □

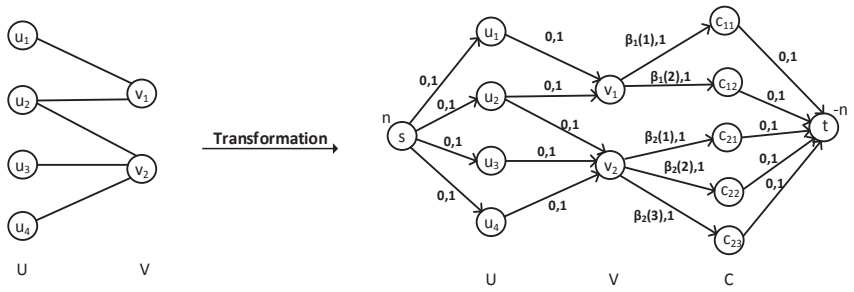

Fig. 1. Transformation to flow network $\mathcal{N}$ with costs. The parameters (cost, capacity) are labelled on each edge. The source s has n units of supply and the sink t has n units of demand.

Noting that $\sum_{i=1}^{M} g_i(0)$ is a constant under any semi-matching M, the following theorem immediately follows from Lemmas 1 and 2.

Theorem 1. *A semi-matching M is optimal for $\mathcal{P}_{\mathbf{g}}^{sum}$ if and only if f^M is a min-cost flow in $\mathcal{N}$.*

Theorem 1 asserts that solving for a min-cost flow in $\mathcal{N}$ is equivalent to solving $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$. In Section III-B, we use this equivalence to derive a simple optimality criterion for $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$, which will be used later to show that an optimal semi-matching for $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$ is also optimal for $\mathcal{P}_{\mathbf{g}}^{\text{max}}$.

Relation between $\mathcal{P}_{\mathbf{g}}^{sum}$ and $\mathcal{P}_{\mathbf{g}}^{max}$: We note that the maximum cost among all edges between V and C that have a unit flow in f^M is equal to the maximum differential cost $\Gamma_{\mathbf{g}}(M)$, as defined in (1). To see this, note that $\Gamma_{\mathbf{g}}(M)$ can also be expressed as follows:

$$\Gamma_{\mathbf{g}}(M) = \max_{v_i \in V} \beta_i(\deg_M(v_i)). \tag{7}$$

The above equation holds because, for each machine vertex v_i, f^M has a unit flow on an edge from v_i to cost center $c_{i\deg_M(v_i)}$, and $\beta_i(\deg_M(v_i))$ is the maximum cost among all the costs on edges outward from v_i that have a unit flow under f^M. Now, given M, the sum-cost objective is simply the sum of edge costs that have unit flow in f^M, and the maximum-differential-cost objective is given by the maximum cost among all edges that have unit flow in f^M. This describes the fundamental relation between $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$ and $\mathcal{P}_{\mathbf{g}}^{\text{max}}$.

In the rest of the paper we use the following terminology. Given a semi-matching M, $\beta_i(\deg_M(v_i))$ is the differential cost of v_i. Further, given a vertex set $S \subseteq V$, we refer to v_i as a *maximum-edge-cost vertex* in S, if the differential cost of v_i is larger than the differential costs of all the vertices in S, under M. If $S = V$, then v_i is a maximum-differential-cost vertex.

B. Optimality for $\mathcal{P}_{\mathbf{g}}^{sum}$

In the following we first define an alternating path in a semi-matching M.

Definition 1. *Given a semi-matching M, an alternating path $\mathbf{p}$ from vertex v_1 to vertex v_r is a sequence of edges given by $\mathbf{p} = \{(v_1, u_1), (u_1, v_2), \ldots, (u_{r-1}, v_r)\}$ such that $(v_k, u_k) \in M, \forall k = 0, \ldots, r-1$.*

Note that the edges $(u_k, v_{k+1}) \notin M$. An important property of an alternating path is that, if we switch the matching and non-matching edges with respect to M along $\mathbf{p}$, we obtain a new semi-matching. The new semi-matching can be formally expressed as the symmetric difference of $\mathbf{p}$ and M, given by $\mathbf{p} \oplus M = (\mathbf{p} \backslash M) \cup (M \backslash \mathbf{p})$. Furthermore, in the new semi-matching $\mathbf{p} \oplus M$, the load on vertex v_1 is reduced by 1 and the load on vertex v_r is increased by 1 while the loads on all other vertices remain the same as in M. In the following we present the definition of a CRP based on the above observations.

Definition 2. *Given a semi-matching M, a Cost Reducing Path is an alternating path $\mathbf{p}$ such that $cost_{\mathbf{g}}(\mathbf{p} \oplus M) < cost_{\mathbf{g}}(M)$.*

By definition, switching the matching and non-matching edges with respect to M along a CRP reduces the sum cost. Since $\mathbf{p}$ is an alternating path, the loads on the vertices under M and $\mathbf{p} \oplus M$ only differ at v_1 and v_r. The criterion for the existence of a CRP from v_1 to v_r can be expressed in terms of the loads on the vertices v_1 and v_r as follows:

$$\begin{aligned} &\text{cost}_{\mathbf{g}}(M) > \text{cost}_{\mathbf{g}}(\mathbf{p} \oplus M) \\ \Leftrightarrow\ & g_1(\deg_M(v_1)) + g_r(\deg_M(v_r)) \\ & > g_1(\deg_M(v_1) - 1) + g_r(\deg_M(v_r) + 1) \\ \Leftrightarrow\ & \beta_1(\deg_M(v_1)) > \beta_r(\deg_M(v_r) + 1). \end{aligned} \tag{8}$$

In other words, a CRP exists from v_1 to v_r if and only if condition (8) is satisfied.

In the following we present some important properties regarding the existence of a CRP, which are direct consequences of the CRP criterion given in (8) and the non-decreasing property of $\beta_i(.)$. Later, these properties will be used extensively in the proofs of our results. Let M be a given semi-matching:

Property 1. *If the differential cost of vertex v_r is at least the differential cost of vertex v_1, i.e., $\beta_r(deg_M(v_r)) \geq \beta_1(deg_M(v_1))$, then there is no CRP from v_1 to v_r.*

Property 2. *Given a vertex set $S \subseteq V$, if $v_{\hat{i}}$ is a maximum-edge-cost vertex in S, then there is no CRP from any vertex in S to $v_{\hat{i}}$.*

Property 3. *If $v_{\hat{i}}$ is a maximum-differential-cost vertex, then there is no CRP from any vertex in V to $v_{\hat{i}}$.*

The following definition will be used extensively in this work.

Definition 3. *Let $\mathbf{p}$ be a CRP under M. We say that $\mathbf{p}$ is cancelled, if matching and non-matching edges with respect to M along $\mathbf{p}$ are switched.*

In the following theorem we characterize the optimality criterion for $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$ based on the notion of CRP.

Theorem 2. *A semi-matching is optimal for $\mathcal{P}_{\mathbf{g}}^{sum}$ if and only if there is no cost-reducing path in it.*

Proof. The proof of the theorem is based on establishing the one-to-one correspondence between a CRP in a semi-matching and a negative cost cycle, in the residual graph of $\mathcal{N}$ with respect to a flow corresponding to the semi-matching. The proof details are given in Appendix VIII-B of [18]. □

We note that the above network equivalence and the optimality result are extensions to the results proved in [14] to accommodate our machine specific cost functions. Accordingly, our proof methodology closely follows [14], except with machine specific edge costs.

Next, we show that the optimality criterion given in Theorem 2 is applicable to $\mathcal{P}_{\mathbf{g}}^{\text{max}}$ as well.

C. Optimality for $\mathcal{P}_{\mathbf{g}}^{max}$

We note that cancelling a CRP in M does not increase the maximum differential cost $\Gamma_{\mathbf{g}}(M)$. We state this in the following lemma.

Lemma 3. *If $\mathbf{p}$ is a CRP in M, then $\Gamma_{\mathbf{g}}(\mathbf{p} \oplus M) \leq \Gamma_{\mathbf{g}}(M)$.*

Proof. The proof is based on the definition of the CRP, and the fact that cancelling a CRP increases the load on one vertex by one and decreases the load on another vertex by one, while all other loads remain unchanged. The proof is given in Appendix VIII-C of [18]. □

Next, we show that a semi-matching that minimizes the sum-cost objective also minimizes the maximum-differential-cost objective.

Lemma 4. *All optimal semi-matchings for $\mathcal{P}_{\mathbf{g}}^{sum}$ have equal maximum differential cost.*

Proof. We prove the lemma by contradiction. In particular, if two optimal semi-matchings for $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$ have unequal maximum differential costs, then we show that there exists a CRP in the optimal semi-matching that has higher maximum differential cost. The proof details are given in Appendix VIII-D of [18]. □

Lemma 5. *There exists an optimal semi-matching for $\mathcal{P}_{\mathbf{g}}^{sum}$ that is also optimal for $\mathcal{P}_{\mathbf{g}}^{max}$.*

Proof. Let $\hat{M}$ be an optimal semi-matching for $\mathcal{P}_{\mathbf{g}}^{\text{max}}$, i.e., $\Gamma_{\mathbf{g}}(\hat{M})$ is the optimal objective value for $\mathcal{P}_{\mathbf{g}}^{\text{max}}$. If there is no CRP under $\hat{M}$, then from Theorem 2, $\hat{M}$ is an optimal semi-matching for $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$. If there are CRPs under $\hat{M}$, then from Lemma 3, cancelling all those CRPs does not increase the maximum differential cost of the resulting semi-matching beyond $\Gamma_{\mathbf{g}}(\hat{M})$. Also, cancelling all those CRPs cannot decrease the maximum differential cost of the resulting semi-matching to below $\Gamma_{\mathbf{g}}(\hat{M})$, since otherwise we would have a contradiction to our assumption that $\hat{M}$ is an optimal semi-matching for $\mathcal{P}_{\mathbf{g}}^{\text{max}}$. Therefore, after cancelling all CRPs, the resultant semi-matching should have its maximum differential cost equal to $\Gamma_{\mathbf{g}}(\hat{M})$. Also, the resultant semi-matching has no CRPs, so from Theorem 2 it is an optimal semi-matching for $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$. □

Theorem 3 is a direct consequence of Lemmas 4 and 5:

Theorem 3. *An optimal semi-matching for $\mathcal{P}_{\mathbf{g}}^{sum}$ is also optimal for $\mathcal{P}_{\mathbf{g}}^{max}$.*

Remark 2: Specialized versions of Theorem 3 have been proven for specific cost functions in the literature. For identical machines, the authors of [14] proved that an optimal semi-matching for sum-cost objective $\text{cost}_M(\mathbf{g})$, where $g_i(\deg_M(v_i)) = \deg_M(v_i)(\deg_M(v_i)+1)/2$ for all v_i, is also optimal for $\Gamma_M(\mathbf{g}) = \max_{v_i \in V} \deg_M(v_i)$. The implication of that result is that, for identical machines, minimizing the sum completion time results in minimizing the makespan. This result was generalized in [6] for solving a data aggregation problem in wireless sensor networks, where

$$g_i(\deg_M(v_i)) = [\frac{s_i}{2}\deg_M(v_i) + q_i](\deg_M(v_i)+1),\ \forall\ v_i,$$

and s_i and q_i are constants specific to a machine (sensor) v_i. They showed that minimizing the sum-cost objective in this case also minimizes $\max_{v_i \in V} s_i \deg_M(v_i) + q_i$. We note that our result is more general and is true for any convex cost function on v_i.

Remark 3: The converse of Theorem 3 is not true. One can find a semi-matching that is optimal for $\mathcal{P}_{\mathbf{g}}^{\text{max}}$ but not optimal for $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$. A counter example can be constructed by first finding an optimal semi-matching for $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$, and then inducing a CRP such that the new semi-matching has the same maximum differential cost as the optimal semi-matching. The new semi-matching will be optimal for $\mathcal{P}_{\mathbf{g}}^{\text{max}}$ but not optimal for $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$. Alternatively, a counter example was given in [14] for identical machines.

IV. Maximum Edge-Cost Cycle Cancelling (MEC^3)

In this section we present the proposed MEC^3 algorithm and analyze its run time complexity.

A. Algorithm Description

MEC^3 is based on the optimality criterion that a semi-matching is optimum if and only if there is no CRP (Theorem 2). Given an initial semi-matching, MEC^3 cancels all the CRPs by enumeration. A CRP can be found by iterating over all machine vertices and checking all alternating paths starting from each machine vertex. However, this exhaustive search

incurs a large amount of redundant work. We incorporate the following improvements in MEC^3. Given a set of machine vertices, search for a CRP starting from a maximum-edge-cost vertex. If no CRP is found, then for further CRP search, the algorithm does not consider the maximum-edge-cost vertex or the machine vertices that have alternating paths from the maximum-edge-cost vertex. We will prove in Section IV-C that no CRP exists starting from those machine vertices. The details of MEC^3 are presented in Algorithm 1.

Algorithm 1: MEC^3 algorithm

1: Find an initial semi-matching M.
2: $S = V$
3: **repeat**
4: Find a CRP $\mathbf{p}$ starting from a vertex $v_{\hat{i}}$, where

$$v_{\hat{i}} \triangleq \arg\max_{v_i \in S} \{\beta_i(\deg_M(v_i))\}.$$

5: **if** a CRP is found **then**
6: $M := \mathbf{p} \oplus M$
7: **else**
8: Remove $v_{\hat{i}}$ and all the vertices in S that have an alternating path from $v_{\hat{i}}$.
9: **end if**
10: **until** S is empty

Given an initial semi-matching, MEC^3 iteratively finds CRPs and cancels them until there is no CRP. Finding a CRP requires $O(|E|)$ time and hence is a time consuming operation. Therefore, finding an initial semi-matching with fewer CRPs can drastically reduce the run time of MEC^3. In our implementation we use the Least-Flexible-Job/Least-Flexible-Machine (LFJ-LFM) rule [19] for this. Our choice is motivated by the demonstrated performance of LFJ-LFM for identical machines [14]. We will also see in Section V that LFJ-LFM performs well in our case. We describe the rule below:

- List the vertices from U in the increasing order of their degrees ("LFJ first").
- For each vertex u_j in the list, do the following:
  1) Assign u_j to v_i if v_i incurs the least cost (computed by $g_i(.)$), over all neighbours of u_j in E.
  2) If the above criterion is satisfied by multiple neighbor vertices of u_j, then assign u_j to the neighbor vertex with the least degree ("LFM first"). Any further ties are broken randomly.

Next, we present the implementation details of MEC^3.

B. Implementation Details

MEC^3 uses the adjacency list representation of G. To find the initial semi-matching using the LFJ/LFM rule, we sort the degrees of the tasks in the ascending order using bucket sort. For each task, we check all the adjacent machines to find the one with the minimum degree. Using the loads assigned to the machines in the initial semi-matching, we build a max heap containing the machine vertices with the key value of a machine vertex equal to the maximum-edge-cost associated with it. Thus, $v_{\hat{i}}$ in line 4 of Algorithm 1 is obtained by extracting the unvisited machine vertex with the maximum key value from the heap.

To find a CRP starting from $v_{\hat{i}}$, we build a Depth-First-Search (DFS) tree of alternating paths rooted at $v_{\hat{i}}$. If a CRP is found to start at a vertex v_i, then the loads of $v_{\hat{i}}$ and v_i are updated. We insert back $v_{\hat{i}}$ into the heap with the new key value. Similarly, we update the key value of v_i in the heap and perform a bubble-up operation to maintain the max heap property. If no CRP is found from $v_{\hat{i}}$, then all the vertices that are visited in the current DFS tree are marked as visited in the heap. The above procedure is repeated until the heap is empty or all vertices in the heap are marked visited.

C. MEC^3 Optimality

In this section we show that the semi-matching output by MEC^3 does not have CRPs. To this end we show that MEC^3 does not miss any CRPs by always starting from $v_{\hat{i}}$ in line 4, and eliminating vertices in line 8, of Algorithm 1.

Let $\mathcal{B}_{\hat{i}}$ be the set of all machine vertices that have an alternating path from $v_{\hat{i}}$. Note that in line 8 of Algorithm 1, we remove $\mathcal{B}_{\hat{i}} \cup \{v_{\hat{i}}\}$ from S. In the following, we us the label (l) in the superscript of the quantities to denote that these quantities are from iteration l of Algorithm 1. The following lemma states that, if in some iteration there is no CRP from $v_{\hat{i}}$, there can be no CRP from the vertex set $\mathcal{B}_{\hat{i}} \cup \{v_{\hat{i}}\}$ in all future iterations.

Lemma 6. *In iteration l of MEC^3, if there is no CRP starting from ${v_{\hat{i}}}^{(l)}$, then there is no CRP starting from any machine vertex in $\mathcal{B}_{\hat{i}}^{(l)} \cup \{{v_{\hat{i}}}^{(l)}\}$, in any iteration $k \geq l$. Furthermore, if $S^{(l)} = V$, then $\Gamma_{\mathbf{g}}(M^{(l)}) = \Gamma_{\mathbf{g}}(M^{(k)})$, for all $k \geq l$.*

Proof. We first prove the result for $k = l$. Assume there is a CRP starting from some vertex ${v_i}^{(l)} \in \mathcal{B}_{\hat{i}}^{(l)}$. The alternating path from ${v_{\hat{i}}}^{(l)}$ to ${v_i}^{(l)}$ appended with the CRP from ${v_i}^{(l)}$ will be a CRP starting from ${v_{\hat{i}}}^{(l)}$, since ${v_{\hat{i}}}^{(l)}$ is the maximum-edge-cost vertex in $S^{(l)}$. By contradiction, the lemma's claim is true for $k = l$.

Since in iteration l no CRP is found starting from ${v_{\hat{i}}}^{(l)}$, the differential cost of ${v_{\hat{i}}}^{(l)}$ does not change. Therefore, in iteration $l+1$ we should have the differential cost of ${v_{\hat{i}}}^{(l+1)}$ less than or equal to the differential cost of ${v_{\hat{i}}}^{(l)}$. Furthermore, if $S^{(l)} = V$, then ${v_{\hat{i}}}^{(l)}$ is the maximum-differential-cost vertex in both $M^{(l)}$ and $M^{(l+1)}$. Therefore, $\Gamma_{\mathbf{g}}(M^{(l)}) = \Gamma_{\mathbf{g}}(M^{(l+1)})$. We have ${v_{\hat{i}}}^{(l+1)} \in S^{(l)} \backslash \{\mathcal{B}_{\hat{i}}^{(l)} \cup \{{v_{\hat{i}}}^{(l)}\}\}$. We claim that no CRP starting from ${v_{\hat{i}}}^{(l+1)}$ will have a vertex from $\mathcal{B}_{\hat{i}}^{(l)} \cup \{{v_{\hat{i}}}^{(l)}\}$ on its path. Clearly, any CRP starting from ${v_{\hat{i}}}^{(l+1)}$ and has ${v_{\hat{i}}}^{(l)}$ on its path implies the existence of a CRP from ${v_{\hat{i}}}^{(l)}$, because the differential cost of ${v_{\hat{i}}}^{(l)}$ is greater than or equal to that of ${v_{\hat{i}}}^{(l+1)}$. Assume some vertex ${v_i}^{(l)} \in \mathcal{B}_{\hat{i}}^{(l)}$ is on a CRP starting from ${v_{\hat{i}}}^{(l+1)}$ to a vertex ${v_r}^{(l+1)}$. Again, the alternating path from ${v_{\hat{i}}}^{(l)}$ to ${v_i}^{(l)}$ appended with the alternating path from ${v_i}^{(l)}$ to ${v_r}^{(l+1)}$ will be a CRP. By contradiction, the claim is

true for $k = l+1$. The same arguments above can be repeated for $k > l+1$. Hence the lemma is proven. □

The optimality of MEC^3 is stated in the following theorem:

Theorem 4. *The semi-matching output by MEC^3 is optimal for both $\mathcal{P}_{\mathbf{g}}^{sum}$ and $\mathcal{P}_{\mathbf{g}}^{max}$.*

Proof. From Lemma 6, we infer that MEC^3 does not miss any CRPs. Whenever a CRP is found, MEC^3 cancels it (line 6 of Algorithm 1). Finally, S becomes empty only when there is no CRP found starting from any machine vertex. As the semi-matching output by MEC^3 has no CRP, its optimality follows from Theorems 2 and 3. □

Next, we use the property of MEC^3 that it always searches for a CRP starting from a maximum-edge-cost vertex to propose another practical improvement that further reduces its run time for solving $\mathcal{P}_{\mathbf{g}}^{\max}$.

D. MEC^3 Run Time Improvement for $\mathcal{P}_{\mathbf{g}}^{max}$

In applications where one aims only at solving $\mathcal{P}_{\mathbf{g}}^{\max}$, the run time of MEC^3 can be improved by imposing an *early stop* condition described as follows: *if no CRP is found starting from v_i in line 4 of Algorithm 1, then stop and output the semi-matching*. In Section V we will show that this early stop condition reduces the run time of MEC^3 significantly while solving $\mathcal{P}_{\mathbf{g}}^{\max}$ for uniform machines. The optimality of MEC^3 with the early stop condition is stated in the following theorem.

Theorem 5. *If $\hat{M}$ is a semi-matching output by MEC^3 with the early stop condition, then $\hat{M}$ is optimal for $\mathcal{P}_{\mathbf{g}}^{max}$.*

Proof. Let l denote the *last* iteration of MEC^3 with the early stop condition. Since MEC^3 stops at the very first time a CRP is not found on line 4 of Algorithm 1, it could not have entered line 8 in any iteration $k \leq l$. Therefore, $S^{(l)} = V$. Let M^* denote the semi-matching output by MEC^3 without the early stop condition, i.e., it continues the iterations beyond l until no CRP is found. Then, from Lemma 6 we have $\Gamma_{\mathbf{g}}(\hat{M}) = \Gamma_{\mathbf{g}}(M^{(l)}) = \Gamma_{\mathbf{g}}(M^*)$. The result follows, as M^* is optimal for $\mathcal{P}_{\mathbf{g}}^{\max}$ (Theorem 3). □

E. MEC^3 Time Complexity

In this section we present the worst-case time complexity analysis for MEC^3.

Lemma 7. *The number of iterations in MEC^3 is upper bounded by $|E|$.*

Proof. The proof is given in Appendix VIII-E of [18]. □

Theorem 6. *The run time of MEC^3 is $O(n^2m^2)$.*

Proof. The run time of MEC^3 is given by $O(\alpha + \eta\gamma)$, where α denotes the run time for finding the initial semi-matching, γ denotes the run time for finding a CRP starting from a vertex in V, and η denotes the number of CRPs cancelled by MEC^3. From Lemma 7, we infer that $\eta \leq |E| \leq nm$. When the LFJ-LFM rule is used to find the initial semi-matching, its time complexity is $O(n \log n + mn)$ [14]. Finding a CRP starting from a vertex v_i can be done by building a depth-first-search tree of alternating paths rooted at v_i. This can be achieved in $O(|E|)$ time. Also, switching matching and non-matching edges along a CRP can be achieved in $O(|E|)$ time. Note that $|E| \leq nm$. Therefore, MEC^3 requires $O(n \log n + mn + nm|E|) = O(n^2m^2)$ time. □

V. Experimental Results

We have implemented MEC^3 and other algorithms in C. The total number of lines in our C files exceeds 3500. We run the programs on a t2.micro instance of Amazon Web Services (AWS), with configuration 2.5 GHz CPU, 1 GB RAM, and Ubuntu. In the following we describe the graph input instances and the objectives that are used for evaluation. We then present the average run time of the algorithms for these input instances and objectives.

A. Graph Instances and Objectives

For problems related to bipartite graphs, the practical performance of different algorithms depend on the structure of the input graphs. In our evaluation we use instances generated from the benchmark graph families described in [20]. These graphs were used to evaluate bipartite matching algorithms in [20], and semi-matching algorithms in [14]. The description of the graph families is given below. All graphs have $|U| = |V|$.

- *FewG and ManyG:* The U-vertices and V-vertices are divided into k groups of equal size. The FewG family has $k = 32$ and the ManyG family has $k = 256$. Each vertex in the jth group of U chooses y random neighbours from the $(i-1)$th, ith, and $(i+1)$th groups (with wrap around) from V, where i is chosen randomly from $\{1, \ldots, |V|\}$ and y is binomially distributed with mean 5.
- *Hilo:* The jth U-vertex is connected to the ith V-vertex for all $\max(1, j-10) \leq i \leq j$.
- *Rope:* The U-vertices and V-vertices are equally partitioned into $t = \frac{n}{6}$ blocks, denoted by $U_0, \ldots, U_{t-1}$ and $V_0, \ldots, V_{t-1}$. The blocks are connected in the following fashion. U_k is connected to V_{k+1} and V_k is connected to U_{k+1}, for all $k \in \{0, \ldots, t-2\}$, and U_{t-1} is connected V_{t-1}. Thus the graph is a "rope" that zigzags between two sides of the graph. Consecutive pairs of blocks along the rope are connected alternately by perfect matchings and random bipartite graphs of average degree 5.
- *Zipf:* The jth U-vertex is connected to the ith V-vertex with probability roughly proportional to $\frac{1}{ij}$. The constants are chosen such that the average degree is 6.

Note that $m = n$ in all the benchmark input instances. We further consider $n \geq m$, and study the run time performance of the algorithms by varying n using problem instances generated as follows. Given m, for each job, the probability that it is linked to a machine is given by q/m. Note that in these problem instances the expected degree of any job is q. We choose $m = 4096$ and $q = 8$. We call this graph family *RandGen*.

In our experiments we assume the uniform machine model. For each machine v_i we choose a s_i value uniformly from $\{1, \ldots, 31\}$. We consider sum completion time, given in (3), as a representative sum-cost objective, and makespan, given in (4), as a representative maximum-differential-cost objective. We compare the run time performance of CANCELALL under both of these objectives. Further, we also use MFMM for run time performance comparison under the makespan objective. The implementation details of CANCELALL and MFMM are given in Appendix VIII-F of [18]. For each data point in the numerical results below, we average over five randomly generated input instances from the chosen graph family.

B. Average Run Time Comparison

For the benchmark input instances, the average run times of MEC^3 and CANCELALL, under the sum-completion-time objective, are presented in Table I. We observe that MEC^3 is much faster than CANCELALL for all benchmark input instances. Note that the run time of CANCELALL is two orders of magnitude longer than the run time of MEC^3.

Under the makespan objective, the run time comparison between MEC^3, CANCELALL, and MFMM is presented in Table II. Again, we observe that MEC^3 is much faster than CANCELALL and MFMM for all benchmark input instances. Note that, under the makespan objective, the run time of CANCELALL is three orders of magnitude longer, and the run time of MFMM is two orders of magnitude longer, than the run time of MEC^3.

Recall that minimizing the sum completion time also minimizes the makespan. Therefore, both MEC^3 and CANCELALL use the cost function in (2) for minimizing the sum completion time thereby minimizing the makespan. As shown in Tables I and II, the run time of CANCELALL remains the same for both these objectives. The drastic reduction in the run time of MEC^3 is due to the early stop condition that we use while solving the makespan minimization problem. This demonstrates the usefulness of the early stop condition in MEC^3.

Finally, in Figure 2 we compare the run time performance of the algorithms for problem instances generated from *RandGen*. An interesting observation is that the increment in the run time of MEC^3, as the number of jobs increases, is much smaller than that of MFMM and CANCELALL.

TABLE I
RUNTIME (SECONDS) COMPARISON FOR THE SUM COMPLETION TIME OBJECTIVE. $n = m = 65536$, AND $\mathbb{E}[s_i] = 16$.

Family	MEC^3	CANCELALL
FewG	0.332	15.02
ManyG	0.353	32.15
Hilo	0.496	13.08
Rope	0.724	34.69
Zipf	0.106	73.87

Discussion: As noted before, using a good initial semi-matching that has a lower number of CRPs is crucial for the run time performance of MEC^3. Recall that we use the LFJ-LFM rule to find an initial semi-matching. From the above results it is evident that LFJ-LFM works well in practice. In our experiments, we have also observed that using random initial semi-matching would drastically lengthen the run time of MEC^3. In contrast to MEC^3, the run time of CANCELALL is not affected by the initial semi-matching. The reason is that, for a given problem instance, the number of recursions in CANCELALL remains roughly the same irrespective of the number of CRPs present in the initial matching.

TABLE II
RUNTIME (SECONDS) COMPARISON FOR THE MAKESPAN OBJECTIVE. $n = m = 65536$, AND $\mathbb{E}[s_i] = 16$.

Family	MEC^3	CANCELALL	MFMM
FewG	0.042	15.02	2.42
ManyG	0.042	32.15	2.40
Hilo	0.075	13.08	3.90
Rope	0.071	34.69	3.15
Zipf	0.057	73.87	2.89

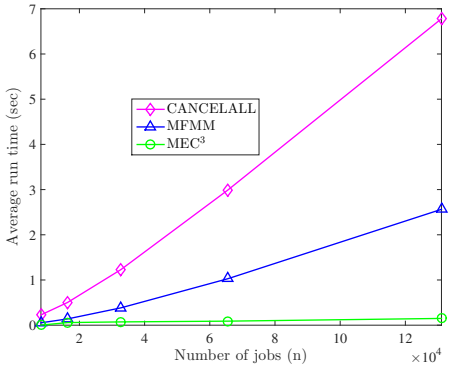

Fig. 2. Runtime comparison for makespan minimization, for $m = 4096$ and expected job degree 8.

VI. Related Work

As mentioned in Section I, the problem of assigning jobs to machines with placement constraints was studied in the literature under various settings and with different objectives [1]–[6], [9], [11]–[13]. In this section, we present further details on the most closely related works in the literature. We categorize these works under the uniform and identical machine models.

A. Uniform Machine Model

The problem of scheduling identical jobs on uniform machines was studied in [9], [11] under the makespan objective. Their network flow transformation method is different and is specific to the makespan objective. Their algorithm, which we call MFMM, requires solving a maximum flow problem iteratively. In Section V, we have shown that MEC^3 is much faster than MFMM. The authors of [11] also studied the problem under the sum completion objective and proposed an algorithm that has $O(n^3 m)$ time complexity.

B. Identical Machine Model

Under this model all machines are identical and all jobs are identical. In [13], the authors studied the problem for the convex cost function $g_i(k) = k^p, p > 0$, for all v_i. They proposed a method that has time complexity $O(n^3m)$. Later, the authors of [14] studied the problem for $g_i(k) = g(k) = k(k+1)/2$ for all i. They solved the problem by transforming it to a min-cost flow problem. An interesting result they proved is that an optimal semi-matching for the sum-cost objective, with $g(k) = k(k+1)/2$ for all v_i, is also optimal with respect to any other convex function $g(.)$. They proposed $\mathcal{A}_{\text{SM2}}$ to solve the min-cost flow problem and proved $O(n^2m^2)$ time complexity. Using benchmark graph input instances, the authors showed that $\mathcal{A}_{\text{SM2}}$ is faster compared with other existing alternatives.

Our work is motivated by the work in [14]. However, we have solved a more general problem with heterogeneous machine cost functions $g_i(.)$. Under such generalization, we prove that minimizing the sum cost implies minimizing the maximum differential cost. The authors of [14] proved this result only for the limited objectives of sum completion time and makespan, and only for identical machines. While both MEC^3 and $\mathcal{A}_{\text{SM2}}$ are based on the principle of cancelling CRPs through enumeration, they are substantially different in the following aspects. First, the criterion for selecting a vertex for finding CRPs is different in MEC^3. Second, to improve speed, MEC^3 removes some vertices if a CRP is not found. Third, MEC^3 incorporates an early stop condition when minimizing the maximum differential cost. Fourth, the arguments used to bound the number of iterations in $\mathcal{A}_{\text{SM2}}$ are not applicable for MEC^3. We have used a novel proof technique to show that the number of iterations in MEC^3 is upper bounded by $|E|$. Finally, MEC^3 requires more sophisticated data structures for implementation due to the generalization in the cost function.

Considering the same problem model as in [14], the authors of [15] proposed a more sophisticated algorithm CANCELALL and proved that it has $O(n^{1.5}m \log n)$ time complexity. To the best of our knowledge, CANCELALL is the only algorithm directly applicable to solve both $\mathcal{P}_{\mathbf{g}}^{\text{sum}}$ and $\mathcal{P}_{\mathbf{g}}^{\text{max}}$, and it has the best worst-case time complexity. However, there is no prior study on its average run time performance. In this work, we have showed that the MEC^3 average run time is two to three orders of magnitude lower than that of CANCELALL.

VII. Conclusion

We have studied the problem of assigning n jobs to m machines, where the jobs have placement constraints, machines are heterogeneous, and the cost incurred at a machine is a general convex function of the number of jobs assigned to the machine. We consider both sum-cost minimization and maximum-differential-cost minimization. We have shown that the former implies the latter. A special case of this result is that minimizing the sum completion time also minimizes the makespan. We propose MEC^3 to solve the sum-cost minimization problem. We further propose an early stop condition for MEC^3 when it is used to solve the maximum-differential-cost minimization problem, which is shown to drastically reduce the run time. We have implemented CANCELALL, an algorithm with the best known worst-case time complexity, and MFMM, a specialized algorithm for makespan minimization. Using benchmark input instances we show that MEC^3 outperforms both algorithms by orders of magnitude, making it an attractive choice for solving the considered problems.

References

[1] C. P. Low, "An efficient retrieval selection algorithm for video servers with random duplicated assignment storage technique," *Information Processing Letters*, vol. 83, no. 6, pp. 315 – 321, 2002.

[2] S. Suri, C. D. Tóth, and Y. Zhou, *Uncoordinated Load Balancing and Congestion Games in P2P Systems*, 2005, pp. 123–130.

[3] S. Suri, D. C. Toth, and Y. Zhou, "Selfish load balancing and atomic congestion games," *Algorithmica*, vol. 47, no. 1, pp. 79–96, 2007.

[4] S. Kittipiyakul and T. Javidi, "Delay-optimal server allocation in multiqueue multiserver systems with time-varying connectivities," *IEEE Transactions on Information Theory*, vol. 55, no. 5, pp. 2319–2333, May 2009.

[5] ——, "Subcarrier allocation in ofdma systems: beyond water-filling," in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on*, vol. 1, Nov 2004, pp. 334–338 Vol.1.

[6] M. Shan, G. Chen, D. Luo, X. Zhu, and X. Wu, "Building maximum lifetime shortest path data aggregation trees in wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 11, no. 1, pp. 11:1–11:24, Jul. 2014.

[7] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03, 2003, pp. 29–43.

[8] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[9] Y. Lin and W. Li, "Parallel machine scheduling of machine-dependent jobs with unit-length," *European Journal of Operational Research*, vol. 156, no. 1, pp. 261–266, 2004.

[10] F. Galčík, J. Katrenič, and G. Semanišin, *On Computing an Optimal Semi-matching*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 250–261.

[11] C.-L. Li, "Scheduling unit-length jobs with machine eligibility restrictions," *European Journal of Operational Research*, vol. 174, no. 2, pp. 1325–1328, 2006.

[12] J. Bruno, E. G. Coffman, Jr., and R. Sethi, "Scheduling independent tasks to reduce mean finishing time," *Commun. ACM*, vol. 17, no. 7, pp. 382–387, Jul. 1974.

[13] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid, "Approximation schemes for scheduling," in *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '97. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997, pp. 493–500.

[14] N. J. A. Harvey, R. E. Ladner, L. Lovász, and T. Tamir, "Semi-matchings for bipartite graphs and load balancing," *J. Algorithms*, vol. 59, no. 1, pp. 53–78, Apr. 2006.

[15] J. Fakcharoenphol, B. Laekhanukit, and D. Nanongkai, "Faster algorithms for semi-matching problems," *ACM Trans. Algorithms*, vol. 10, no. 3, pp. 14:1–14:23, May 2014.

[16] V. B. Cherkassky and V. A. Goldberg, "On implementing the push—relabel method for the maximum flow problem," *Algorithmica*, vol. 19, no. 4, pp. 390–410, 1997.

[17] K. Lee, J. Y.-T. Leung, and M. L. Pinedo, "Scheduling jobs with equal processing times subject to machine eligibility constraints," *Journal of Scheduling*, vol. 14, no. 1, pp. 27–38, 2011.

[18] J. P. Champati and B. Liang, "Efficient minimization of sum and differential costs on machines with job placement constraints (technical report)," 2017. [Online]. Available: http://www.comm.utoronto.ca/%7eliang/publications/techreport/INFOCOM2017TechRepMEC3.pdf

[19] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Springer Publishing Company, Incorporated, 2008.

[20] B. V. Cherkassky, A. V. Goldberg, P. Martin, J. C. Setubal, and J. Stolfi, "Augment or push: A computational study of bipartite matching and unit-capacity flow algorithms," *J. Exp. Algorithmics*, vol. 3, Sep. 1998.