Reoptimization of the Minimum Total Flow-Time Scheduling Problem*

Guy Baram† Tami Tamir†

Abstract

We consider power-aware *reoptimization* problems arising in production planning. Due to unexpected changes in the environment (out-of-order or new machines, modified jobs' processing requirements, etc.), the production schedule needs to be modified. That is, jobs might be migrated from their current machine to a different one. Migrations are associated with a cost – due to relocation overhead and machine set-up times. The goal is to find a good modified schedule, with a low transition cost from the initial one. We consider the objective of minimizing the total flow-time, denoted in standard scheduling notation by $P||\sum C_j$.

We study two different problems: (i) achieving an optimal solution using the minimal possible transition cost, and (ii) achieving the best possible schedule using a given limited budget for the transition. We present optimal algorithms for the first problem and for several classes of instances of the second problem.

1 Introduction

This work studies a power-aware reoptimization variant of the classical scheduling problem of minimizing the total flow-time (denoted in standard scheduling notation by $P||\sum C_j$ [15]). The minimum total flow-time problem can be solved efficiently by the simple greedy SPT rule [28, 8] that assigns the jobs in nondecreasing order by their length. This algorithm, as many other algorithms for combinatorial optimization problems, solves the problem from scratch, for a single arbitrary instance without having any constraints or preferences regarding the required solution - as long as it achieves the optimal objective value. However, many of the real-life scenarios motivating these problems involve systems that change dynamically over time. Thus, throughout the continuous operation of such a system, it is required to compute solutions for new problem instances, derived from previous instances.

Moreover, since there is some cost associated with the transition from one solution to another, a natural goal is to have the solution for the new instance *close* to the original one (under certain distance measure).

Solving a *reoptimization* problem involves two challenges:

1. *Computing* an optimal (or close to the optimal) solution for the new instance.
2. Efficiently *converting* the current solution to the new one.

*A preliminary version of this paper appears in the proceedings of the 1st Mediterranean Conference on Algorithms (MedAlg) December 2012, Ein-Gedi, Israel.

†School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. E-mail: guy.baram@gmail.com, tami@idc.ac.il.

Each of these challenges, even when considered alone, gives rise to many theoretical and practical questions. Obviously, combining the two challenges is an important goal, which shows up in many applications.

The reoptimization problem of minimizing the total flow-time arises naturally in production planning. Due to unexpected changes in the environment (out-of-order or new machines, timetables of task processing, etc.), the production schedule needs to be modified. Rescheduling tasks is costly, due to relocation overhead and machine set-up times. The goal is to find a new feasible schedule, which is as close as possible to the previous one.

Applications: As mentioned above, the scenario we consider often arises in manufacturing systems. In fact, our work is relevant to any dynamic scheduling environment, in which migrations of jobs are associated with a transition cost, energy-loss, or overhead caused due to the need to absorb the migrating jobs. We describe below several less intuitive applications in cloud computing and semiconductor wafers production line.

Consider an RPC (Remote Procedure Call) service. In this environment, a cloud of servers can provide service to a limited number of simultaneous users. If the number of requests is high, another virtual server could be temporarily rented, where the cost for using it is per user. The options are to put the RPC in a queue, thus causing latency in the service, or renting more virtual servers, enabling faster service and paying the additional servers' cost. In this application, the transition cost is not due to the migration itself, but due to the activation cost of the additional resources.

Some of our results will be extended to consider modifications that occur after the processing has begun, that is, at time $t > 0$. For this extension (see Section 2.2) we distinguish between environments in which the currently processed jobs can migrate and be restarted on different machines, and applications in with restarts are not allowed, and a currently processed job must complete its partial processing. The following application describes a system in which restarts are not allowed: In a semiconductor wafers production line, some of the coating methods involve purely physical processes such as high temperature vacuum evaporation (physical vapor deposition - PVD). During the process, a vacuum is created to enable the coating. Once the elements are in a vacuum environment, the process cannot be stopped as if the machine halts, it will be severely damaged [21]. Assume that at time $t > 0$ machines are added. Transferring jobs is costly - to capture the transition overhead and the changes required in programming the machines workplan. Also, the elements that are currently produced, that are already in vacuum state, must complete their production.

We note that our results for change at $t = 0$ are not necessarily static, as in many systems, the same workplan is repeated periodically (daily, etc.), thus, every period can be considered as a schedule starting at time $t = 0$. A change in the fixed periodic schedule is equivalent to a change at time $t = 0$.

1.1 Problem Statement and Notation

An instance of our problem consists of a set J_0 of n_0 jobs and a set of m_0 identical machines. Denote by p_j the processing time of job j. A schedule S_0 of the initial instance is given. That is, for every job in J_0, it is specified on which machine it is assigned and on which time interval it is going to be processed. At any time, a machine can process at most one job and a job can be processed by at most one machine.

At time $t \geq 0$, a change in the system occurs. Possible changes include addition or removal of machines and/or jobs, as well as modification of processing time of jobs in J_0. Let J denote the modified set of jobs, and let $n = |J|$. Let M denote the modified set of machines, and let $m = |M|$. Our goal is to suggest a new schedule, S, for the modified instance, with good objective value and small transition cost form S_0. Assignment of a job to a different machine in S_0 and S is denoted *migration* and is associated with a cost. Formally, we are given a *price list* $\theta_{ii'j}$, such that it costs $\theta_{ii'j}$ to migrate job j from machine i to machine i'. We consider two problems:

1. Rescheduling to an optimal schedule using the minimal possible transition cost.
2. Given a budget B, find the best possible modified schedule that can be achieved without exceeding the budget B.

Some of our results assume unit transition costs, that is, for all j and $i \neq i'$, $\theta_{ii'j} = 1$.[1]

For a given schedule, let C_j be the completion time of job j, that is, the time when the process of j completes. In this work we consider the problem of minimizing the sum of completion times, denote by $\sum C_j$ and also known as *total flow-time*. In the reoptimization problem, given J, J_0, the goal is to find a good schedule for J that is close to the initial schedule S_0.

Example 1: Assume that six jobs of lengths $1, \dots, 6$ are scheduled on a single machine in an optimal SPT order. Assume that a second machine is added, and that all migrations have unit transition cost. Figure 1(a) presents an optimal modified schedule, for which the total flow-time is $\sum C_j = 34$. Migrating jobs appear in grey. The budget required to reach this schedule (or any other schedule with $\sum C_j = 34$) is 3. For a given budget, $B = 2$, it is possible to move, for example, to the modified schedules given in Figures 1(b) and (c), having total flow-time 36 and 35, respectively. The schedule (c) is optimal for this budget.

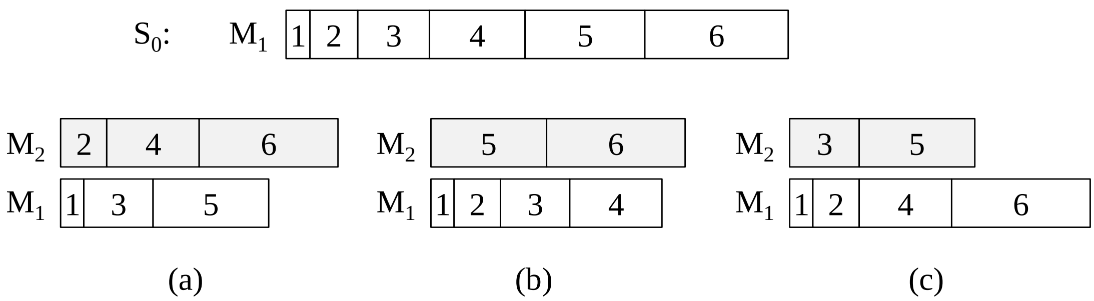


Figure 1: (top) An initial assignment, (a) an optimal reassignment achieved with transition cost 3, (b) a possible, and (c) an optimal reassignments achieved with limited budget $B = 2$.

Example 2: Figure 2(a) presents an optimal SPT schedule of 12 jobs of on three machines. Assume that a forth machine is added. Figure 2(a) presents an optimal modified schedule, for which the total flow-time is $\sum C_j = 55$. The budget required to reach this schedule (or any other schedule with $\sum C_j = 55$) is 4. An important observation is that even though the number of jobs on the added machine is 3, and the initial schedule is optimal, it is not possible to move to an optimal schedule with budget $B = 3$. An optimal solution for $B = 3$ is presented in Figure 2(c). It has total flow-time 56.

[1]Note that the constant 1 can be replaced by any other constant.

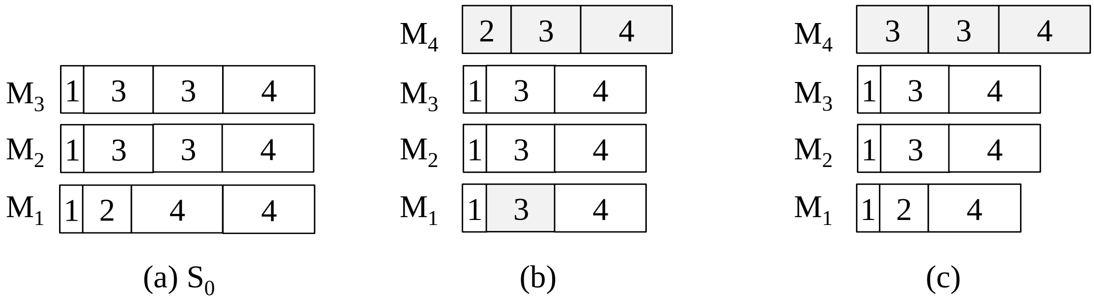


Figure 2: (a) An initial assignment, (b) an optimal reassignment requires transition cost 4, and (c) an optimal reassignment for $B = 3$.

Natural Heuristics: The above examples demonstrate some of the challenges in solving our reoptimization problems, and the fact that simple natural heuristics do not solve the problem optimally, even if all migrations have unit transition cost. Note first that the natural greedy approach of migrating the long jobs if the budget is low (as in Example 1(b)) is sub-optimal. Another natural approach is prefix-SPT – use the budget to maximize the prefix of the schedule that agrees with an SPT schedule. This approach fails on Example 1 (jobs of lengthes 2 and 4 will be migrated, resulting in $\sum C_j = 37$). Moreover, prefix-SPT as well as suffix-SPT (maximize the suffix of the schedule that agrees with an SPT schedule) are not well-defined since the tie-breaking rule, in case of multiple jobs having the same processing time, is crucial - as demonstrated in Example 2.

1.2 Related Work

The 'single-shot' minimum total flow-time, $P||\sum C_j$, can be solved in polynomial time by using the shortest processing time (SPT) rule [28, 8]. The problem is solvable also on unrelated machines, $R||\sum C_j$, by a reduction to a minimum-weight complete matching problem [6, 17].

The work on reoptimization problems started with the analysis of *dynamic graph* problems (see e.g. [11, 30] and a survey in [9]). These works focus on developing data structures supporting update and query operations on graphs. Reoptimization algorithms were developed also for some classic problems on graphs, such as shortest-path [23, 22] and minimum spanning tree [1].

A different line of research deals with the computation of a good solution for an NP-hard problem, given an optimal solution for a close instance. Among the problems studied in this setting are TSP, [4, 5], Steiner Tree on weighted graphs [12] and Knapsack [2]. A survey of other research in this direction is given in [3]. In all of the above works, the goal is to compute an optimal (or approximate) solution for the modified instance. The resulting solution may be significantly different from the original one, since there is no cost associated with the transition among solutions.

The paper [26] suggests the framework we adopt for this work, in which the solution for the modified instance is evaluated also with respect to its difference from the initial solution. This framework is in use also in [25], to analyze algorithms for data placement in storage area network. Considering both the quality of the solution and the transition cost from an initial solution can also be seen as a special case of *multi-objective* optimization problems. In these problems, there are several weight functions associated with the input elements. The goal is to find a solution whose quality is measured with respect to a combination of these weights (see e.g., [24, 16]).

Other related work consider several graph algorithms that we apply or adjust in this work. A matching $M \subseteq E$ in a graph $G = (V, E)$ is set of edges such that each node in V appears in at most one edge in M. A Bipartite graph $G = (V, E)$ is a graph in which the vertex set V can be divided into two disjoint subsets V_1 and V_2 such that $E \subseteq V_1 \times V_2$. A complete matching in a bipartite is a matching of size $\min(|V_1|, |V_2|)$. A min-weight complete matching can be found using the hungarian method [20] or the Push-relabel algorithm [14]. Their runtime is $O(\sqrt{|V|}|E|\frac{\log(|V|^2/|E|)}{\log|V|})$ [19]. Another problem whose solution we use as a black box, is the problem of finding a min-cost max-flow in a network. Different approaches have been proposed for solving the min-cost max-flow problem. The classical algorithms for min-cost max-flow problems are the algorithm of Ford-Fulkerson [13] and the minimum-cost augmentation method [10]. The minimum mean-cost cycle-canceling algorithm, developed by Goldberg and Tarjan, is a strongly polynomial time algorithm for min-cost max-flow [14]. Its running time is $O(|V|^2|E|^3 \log|V|)$ [19].

1.3 Our Results

In Section 2 we explore the problem of moving to a modified optimal schedule using the minimal required budget. We present optimal algorithms that return both an optimal schedule and the minimal budget B required to reach an optimal schedule. We first describe an optimal algorithm for arbitrary migration costs and arbitrary changes in the instance. Its running time is dominated by the time required to find a minimum weight complete matching in a complete bipartite graph with $O(nm)$ vertices. We then present a more efficient algorithm for instances with unit migration costs. The time complexity of this algorithm varies between $O(n)$ (if the initial schedule is optimal) and $O(n \log n)$ (for arbitrary initial schedule). The first algorithm is described assuming the modification takes place at time $t = 0$. In Section 2.2 we describe how and under which conditions it can be extended to handle modifications at time $t > 0$. The second algorithm is valid for changes at any time $t \geq 0$. In Section 2.4, we characterize instances for which it is possible to solve the problem by a simple linear-time algorithm.

In section 3 we consider the problem of rescheduling with a limited budget. The goal is to utilize the budget in the best possible way, that is, the modified schedule should have a low total flow-time - the minimal possible among all schedules that can be achieved using the given budget. Our results for this model assume unit migration costs, thus, the budget B gives the maximal number of allowed migrations. We present optimal algorithms for two cases: when the budget is a constant and when migrations are allowed only to new machines.

We conclude, in Section 4, with a discussion and some directions for future work. We note that our results can be applied also on a sequence of modifications. That is, the environment might change more than once, and the algorithms are performed after each modification.

2 Optimal Modified Schedule Using Minimal Budget

In this section we consider the problem of moving to a modified optimal schedule with respect to the minimal total flow-time objective using the minimal required budget.

2.1 Arbitrary Costs and Modifications at time $t = 0$

Let S_0 be a given initial schedule. We do not assume that S_0 is optimal nor that it has a specific structure or properties. Assume that at time $t = 0$, the environment is modified. Possible modifications include addition or removal of machines and/or jobs, and changes in jobs' processing times. The price-list $\theta_{ii'j}$ specifies for every job j assigned to machine i, how much it costs to migrate j to machine i'. The goal is to find a new schedule, S, which is optimal with respect to the total flow-time, and has the minimal transition cost from S_0 among all optimal schedules.

We reduce the problem into a minimum weight complete matching problem in a bipartite graph. This approach was suggested by Horn [17], and Bruno, Coffman and Sethi [6] for solving the problem of minimum flow-time on unrelated machines ($R||\sum C_j$). While the processing time of the jobs do not change due to migrations, it is possible to adopt this technique for our problem by setting the weights in the corresponding bipartite graph in a way that reflects the migration overhead.

Recall that n and m represent the number of jobs and machines in the modified instance. Let $G = (V, E)$, where $V = J \cup U$. The set J represents the set of n jobs (a single node per job). The set U consists of mn nodes, q_{ik}, for $i = 1, \ldots, m$ and $k = 1, \ldots, n$, where node q_{ik} represents the k^{th} from last position on machine i. The edge set E includes an edge (v_j, q_{ik}) for every node in J and every node in U (a complete bipartite graph). The following is an optimal algorithm for our problem. Note that edge weights (determined in Step 2) consist of two components: a dominant component (with large factor Z) corresponding to the contribution of a job assigned in a specific position to the total flow-time, and a minor component corresponding to the associated transition cost. Both components are combined to form a single weight. Figure 3 illustrates the bipartite and the edges corresponding to a single job.

Algorithm 1 - An optimal algorithms for rescheduling using minimal budget

---

1. Let $\theta_{ii'j}$ be a price list, i.e., it costs $\theta_{ii'j}$ to migrate job j from machine i to machine i'. In particular, for all i, j, $\theta_{iij} = 0$.

   Let $\Delta = \max_{j,i,i'} \theta_{ii'j}$, and let Z be a constant lager than $n\Delta$.

2. Let G be the complete bipartite graph corresponding to the problem. Set the edge weights $w : E \rightarrow \mathbb{R}$ as follows:

   - For every job that is assigned to i, let $w(v_j, q_{ik}) = Zkp_j$.
   - For every $i' \neq i$, let $w(v_j, q_{i'k}) = Zkp_j + \theta_{ii'j}$.

3. Find a min-weight complete matching in G. Let H denote the set of edges in this matching.

4. Return the schedule corresponding to H. That is, for every $(v_j, q_{i',k}) \in H$, assign j in the k^{th} from last position on machine $M_{i'}$. The minimum transition cost is $\sum_{(v_j, q_{i',k}) \in H} \theta_{ii'j}$, where i is the machine on which j is assigned in S_0.

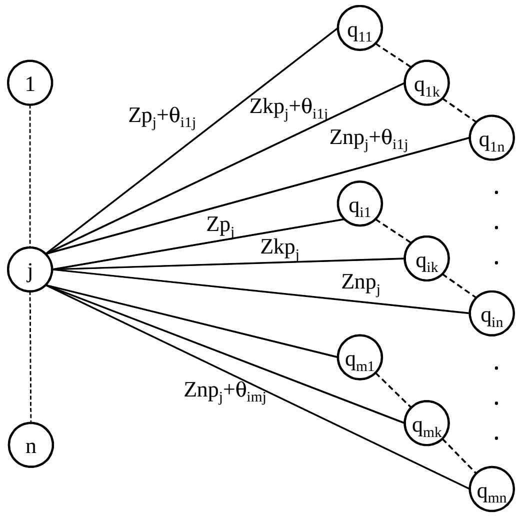


Figure 3: The bipartite graph for Algorithm 1. The job j is assigned to machine i in S_0.

Theorem 2.1 *Algorithm 1 returns an optimal schedule using the minimal possible transition cost from S_0.*

Proof: The proof consists of two claims, the first concerning the optimality with respect to the total flow-time, and the second concerning the optimality with respect to the transition cost.

Consider the modified instance and assume that transitions are not associated with costs. We get a simple $P||\sum C_j$ problem on the modified set of machines. While it is possible to solve the problem by SPT rule, it is also possible to solve the corresponding $R||\sum C_j$ problem on unrelated machines assuming that for any machine i and job j it holds that $p_{i,j} = p_j$. Let G' denote the bipartite graph built for solving the corresponding min-weight matching problem [17, 6]. The graph G' has the same vertex-set and the same edge-set as the graph G built in Algorithm 1. The graphs G and G' differ in the edge weights. It is well known that if job j is the k^{th} from the last job to run on M_i, it contributes exactly k times its processing time to the sum of completion times. Therefore, in G', the weight $w'(v_j, q_{ik})$ of an edge (v_j, q_{ik}) is simply kp_j. Since we assume no transition costs, these weights are independent of the machine i.

Claim 2.2 *The set of edges H found in step 3, induces a feasible schedule S with minimum total flow-time.*

Proof: We show that the set of edges H is a minimum weight matching also in G' - and thus, as shown in [17, 6], it corresponds to a feasible schedule with minimum total flow-time. First, since G and G' differ only in edge weights, H is a legal matching in G'. Assume by way of contradiction that H is not minimal with respect to the weights w' in G', and that H^* is a complete matching in G' with a lower weight. Therefore,

$$1 + \sum_{e \in H^*} w'(e) \le \sum_{e \in H} w'(e). \tag{1}$$

By definition of w and w', the weight of H^* in G is

$$\sum_{e \in H^*} w(e) = Z \sum_{e \in H^*} w'(e) + \sum_{e=(v_j, q_{ik}) \in H^*} \theta_{ii'j}.$$

By definition of Z, it holds that $\sum_{e=(v_j,q_{ik})\in H^*} \theta_{ii'j} \leq n\Delta < Z$. Therefore,

$$\sum_{e\in H^*} w(e) < Z(1 + \sum_{e\in H^*} w'(e)). \tag{2}$$

Since H is a min-weight matching with respect to the weights w, it holds that

$$\sum_{e\in H} w(e) \leq \sum_{e\in H^*} w(e). \tag{3}$$

Combining Equations (1) (multiplied by Z),the definitions of w and w', (3) and (2), we get the following contradiction:

$$Z(1 + \sum_{e\in H^*} w'(e)) \leq Z \sum_{e\in H} w'(e) \leq \sum_{e\in H} w(e) \leq \sum_{e\in H^*} w(e) < Z(1 + \sum_{e\in H^*} w'(e)).$$

$\blacksquare$

We conclude that the schedule S returned by the algorithm is a feasible schedule minimizing the total flow-time, and turn to show it also minimizes the transition cost from S_0.

Claim 2.3 *Among all schedules achieving minimum total flow-time, the schedule S induced by H has the minimal transition cost from S_0.*

Proof: Let H^* be any perfect matching in G, corresponding to a schedule, S^*, achieving minimum total flow-time. We show that the transition cost from S_0 to S^* is not lower than the transition cost to S. We know that H is a min-weight complete matching in G, therefore,

$$\sum_{e\in H} w(e) \leq \sum_{e\in H^*} w(e). \tag{4}$$

Also, since both induce schedules achieving minimum total flow-time and the weights w' in G' reflect the total flow-time without the transition costs,

$$\sum_{e\in H} w'(e) = \sum_{e\in H^*} w'(e). \tag{5}$$

The definition of w implies that for every matching $\hat{H}$, it holds that

$$\sum_{e\in \hat{H}} w(e) = Z \sum_{e\in \hat{H}} w'(e) + \sum_{e=(v_j,q_{ik})\in \hat{H}} \theta_{ii'j}, \tag{6}$$

where the second term is exactly the transition cost from the initial schedule to the schedule induced by $\hat{H}$. Therefore, by applying Equation (6) on both H and H^*, and using Equations (5) and (4), we get:

$$\sum_{e=(v_j,q_{ik})\in H^*} \theta_{ii'j} - \sum_{e=(v_j,q_{ik})\in H} \theta_{ii'j} = \sum_{e\in H^*} w(e) - \sum_{e\in H} w(e) \geq 0.$$

We conclude that the transition cost to S^* is not lower than the transition cost to S. $\blacksquare$ $\blacksquare$

2.2 Modification Occurs at time $t > 0$

In this section we extend the algorithm to consider systems that are modified after the processing has begun, that is, at time $t > 0$. Denote by J_t the set of jobs processed at time t, and let, for every machine i, $\gamma_i \geq 0$ denote the time required to complete the job from J_t processed at time t on machine i. As detailed in the introduction, in some systems, the processing of a job $j \in J_t$ must complete on its current machines. In other systems, currently processed jobs can be migrated to another machine. We present different algorithms for the two settings.

2.2.1 Restarts are not allowed

When restarts are not allowed, the modification of machines' removal, is not possible - if machines can be removed, and restarts are not allowed then the problem is not well-defined for the jobs that are currently processed. Thus, we assume that the modifications are machines' addition and/or changes in the set or processing times of jobs. The goal is to determine the schedule of jobs whose processing did not begin before time t. An optimal algorithm for this case is based on the observation that for the modified schedule machine i is available starting at time γ_i. Algorithm 1 can be generalized by setting the weights $w_{noR} : E \to \mathbb{R}$ in the bipartite graph (determined in Step 2) in the following way:

- For every job that is assigned to M_i, let $w_{noR}(v_j, q_{ik}) = Z(kp_j + \gamma_i)$.
- For every $i' \neq i$, let $w_{noR}(v_j, q_{i'k}) = Z(kp_j + \gamma_{i'}) + \theta_{ii'j}$.

Since restarts are not allowed, the only difference from the case in which the modification occurs at time $t = 0$ is the fact that machine i is available only from time γ_i. Thus, if job j is the k^{th} from the last job to run on M_i, it contributes exactly γ_i plus k times its processing time to the sum of completion times. As in Algorithm 1, the weights consist of a dominant component (with large factor Z) ensuring that schedule achieves minimum total flow-time, and a minor component ensuring the minimal possible transition cost.

The availability time of machine i is added to the dominant component, as it affect the flow-time of the jobs assigned to it. The proof of the following theorem follows directly the proof of Theorem 2.1.

Theorem 2.4 *Algorithm 1 with weights w_{noR} returns an optimal schedule using the minimal possible transition cost from S_0, when restarts are not allowed.*

2.2.2 Restarts are allowed

When restarts are allowed, a job $j \in J_t$ might complete its processing on its current machine, but can also migrate to a different machine. If migrated, the corresponding transition cost is applied and the job must *restart*. We assume that preemptions are not allowed[2]. Another possibility for a job $j \in J_t$ is to remain on its current machine, but delay its processing - letting jobs migrating from other machines precedes it. In this case, the job must restart, but no transition cost is applied, as no migration is performed.

Recall that for every machine i, $\gamma_i \geq 0$ denotes the time required to complete the job from J_t processed at time t on machine i. Our algorithm assumes that the initial schedule, S_0, was

[2]Enabling preemptions affects all the jobs of the instance, thus causing the problem to be intractable [27].

optimal and that the modification includes machines' addition. Algorithm 1 can be generalized by setting the weights $w_R : E \to \mathbb{R}$ in the bipartite graph (determined in Step 2) in the following way:

- For every job $j \in J_t$ that is currently processed on M_i let $w_R(v_j, q_{ik}) = Zk\gamma_i$.
- For every job $j \notin J_t$ that is assigned to M_i, let $w_R(v_j, q_{ik}) = Zkp_j$.
- For every $i' \neq i$, let $w_R(v_j, q_{i'k}) = Zkp_j + \theta_{ii'j}$.

Note that the processing time of the currently processed job j on i is assumed to be γ_j even though j might not be assigned to be the first job on M_i. The next lemma justifies these settings.

Lemma 2.5 *For every machine i, let $j \in J_t$ be the job processed by M_i at time t, then there is no optimal schedule in which j is in round 2 or more on M_i.*

Proof: Assume that there exists an optimal reschedule S^* in which j is scheduled second (or later) on the machine M_1 on which it was assigned in the initial configuration. Recall that γ_1 is the remaining processing time of j on M_1. Since in any optimal schedule, the jobs on every machine are sorted from shorter to longer, there must be at least one job, j', not longer than γ_1, before j. If $\gamma_1 = p_{j'}$ then by swapping j and j', the value of the total flow-time is unchanged and j is assigned first as required. Thus, we assume that $p_{j'} < \gamma_1$. Since we assume that the initial schedule is SPT, and j was processed at time t by M_1, the job j' was assigned on a different machine at time t. Given that two jobs from J_t are on M_1 in S^*, by the pigeonhole-principle, there must be a machine, M_2, on which no job from J_t is assigned in S^*. Let j^* be the first job on M_2 in S^* (see Figure 4(a)). Since $j^* \notin J_t$ it must be that $p_{j^*} \geq \gamma_1 > p_{j'}$. Denote by k_1 and k_2 the number of jobs on M_1 and M_2 in S^*, respectively.

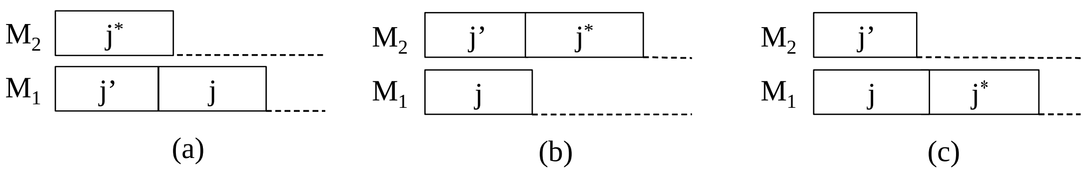


Figure 4: (a) The assumed schedule, (b) A better schedule if $k_1 > k_2$, and (c) a better schedule if $k_1 \leq k_2$.

If $k_1 > k_2$, move j' to be first on machine M_2 (see Figure 4(b)). The contribution of j' to the total flow-time before the migration is $k_1 p_{j'}$. The contribution of j' to the total flow-time after the migration is $(k_2+1)p_{j'}$. For any $k_2 < k_1$, this migration does not increase the total flow-time. Moreover, it might save the transition cost of j' (if its original machine is M_2), thus, the resulting schedule is either better than S^*, contradicting its optimality, or has the same total flow-time and transition cost as S^*, and it satisfies the requirement that j is first on M_1.

If $k_1 \leq k_2$, move j and j^* to be the first and second jobs on M_1, and move j' to be the first job on M_2 (see Figure 4(c)). The contribution of these three jobs to the total flow-time before the change is $k_1 p_{j'} + (k_1 - 1)\gamma_1 + k_2 p_{j^*}$. The contribution of these three jobs to the total flow-time after the change is $k_1\gamma_1 + (k_1 - 1)p_{j^*} + k_2 p_{j'}$. The total flow-time reduced by $(p_{j^*} - p_{j'})(k_2 - k_1) + p_{j^*} - \gamma_1$, which is positive for any $p_{j^*} \geq \gamma_1 > p_{j'}$ and $k_2 \geq k_1$. Thus, the resulting schedule has lower total flow-time, contradicting the fact that S^* is optimal. ■

We conclude that Algorithm 1 with the weights w_R solves optimal the reoptimaization problem with modifications at time $t > 0$ and restarts allowed. Note that the output of the algorithm is an SPT schedule, therefore, the algorithm can also handle a sequence of modifications.

2.3 An Efficient Algorithm for Identical Migration Costs

In this section we consider systems with identical migration costs, that is, for all j, i, i', it holds that $\theta_{j,i,i'} = \theta$. We present an efficient algorithm for finding an optimal modified schedule using the minimal possible budget. The algorithm can be applied for addition or removal of machines and/or jobs, as well as changes in jobs' processing times.

The algorithm is based on some properties of the SPT algorithm [28, 8] for $P||\sum C_j$. For completeness, we describe a specific form of SPT algorithm: Given an instance of n jobs and m parallel machines, add dummy jobs of length 0 such that the total number of jobs is a multiple of m. Specifically, if n is not a multiple of m, then add to the instance $m - (n \bmod m)$ jobs of length 0. The dummy jobs can be scheduled on arbitrary machines and (when rescheduling) their migration cost is 0. Given that n is a multiple of m, the SPT algorithm can be described as follows: First, sort the jobs in non-decreasing order of processing time (break ties arbitrarily). Next, partition the jobs into n/m *rounds* of m jobs each. The k-th round consists of the jobs indexed $(k-1)m+1, \ldots, km$ in the sorted list. Schedule on each machine one job from the first round, followed by one job from the second round, etc.

We use the following known property of SPT schedules: *the internal assignment of jobs from a particular round to the machines does not affect the total flow-time*. That is, any schedule in which the m jobs of round k are assigned on the k-th slots of the m machines is optimal.

Let L be the set of job-lengths in the modified instance. The set L includes at most n distinct values. By the above property of SPT schedules, an optimal schedule can be characterized by the numbers $n_{\ell,k}$, for all $\ell \in L$ and $1 \leq k \leq \frac{n}{m}$, where $n_{\ell,k}$ is the number of jobs of length ℓ in round k, in any optimal schedule. Moreover, the problem of finding an optimal schedule using minimum transition cost reduces to the problem of finding a schedule obeying the optimal $n_{\ell,k}$ values with a minimal number of migrations from the initial schedule. The following is an overview of our optimal algorithm:

Algorithm 2 - An efficient optimal algorithm for rescheduling with identical migration costs.

---

1. For every length $\ell \in L$ and round $1 \leq k \leq \frac{n}{m}$, calculate $n_{\ell,k}$, the number of jobs of length ℓ in round k, in any optimal modified schedule.

2. Partition L into two sets of job lengths: Let $L_1 \subseteq L$ be the set of lengths such that $\ell \in L_1$ if and only if $n_{\ell,k} > 0$ for a single round k. Let $L_2 = L \setminus L_1$ be the set of lengths such that $\ell \in L_2$ if and only if $n_{\ell,k} > 0$ for more than a single round.

3. For every round $1 \leq k \leq \frac{n}{m}$, schedule a maximal number of non-migrating jobs in round k. First, assign jobs having lengths in L_1, then in L_2. When assigning jobs from L_2, give higher priority to short jobs.

4. Schedule migrating jobs.

The idea is to assign first a maximal number of non-migrating jobs, and then assign the migrating jobs. When assigning the non-migrating jobs, we first assign the more restricted jobs – having lengths in L_1, and must be assigned in a specific round, and then the more flexible jobs whose lengths are in L_2 (and can be assigned in more than one specific round).

Denote by S the schedule built by the algorithm. Steps (3-4) are implemented as follows: Denote by $S_{i,k}$ the slot in the k^{th} round on machine i. Initially, for all $1 \leq i \leq m, 1 \leq k \leq \frac{n}{m}$ it holds that $S_{i,k}$ is available ($= \emptyset$). During steps (3-4) some slots are assigned to non-migrating jobs. Whenever a job j of length ℓ is assigned to the k-th slot on machine i, the corresponding variable $S_{i,k}$ is set to j, and the corresponding counter of $n_{\ell,k}$ is reduced by one. Specifically, steps (3-4) are implemented as follows:

Step 3: Step 3 consists of $\frac{n}{m}$ iterations. In iteration k, the algorithm assigns non-migrating jobs into slots of round k. Consider a slot $S_{i,k}$. Let $ForFree(i,k)$ denote the set of jobs that can be assigned to $S_{i,k}$ with no migration. Formally, $j \in ForFree(i,k)$ if and only if (i) $n_{p_j,k} > 0$, (ii) j is assigned to M_i in S_0, and (iii) j was not assigned to M_i in earlier rounds.

In step 3, if possible, the algorithm assigns to $S_{i,k}$ a job from $ForFree(i,k)$ giving priority to lengths in L_1, and then to shorter lengths in L_2. Formally,

For $k = 1$ to $\frac{n}{m}$
  For $i = 1$ to m
    Calculate $ForFree(i,k)$.
    If $ForFree(i,k) \neq \emptyset$
      If there exists $j \in ForFree(i,k)$ such that $p_j \in L_1$. Set $S_{i,k} = j$, $n_{p_j,k} = n_{p_j,k} - 1$.
      Else, let j be the shortest job in $ForFree(i,k)$ such that $p_j \in L_2$.
        Set $S_{i,k} = j$, $n_{p_j,k} = n_{p_j,k} - 1$.

Step 4: Step 4 consists of $\frac{n}{m}$ iterations. In iteration k, the algorithm assigns, with migrations, jobs to slots $S_{i,k}$ for which $ForFree(i,k) = \emptyset$. Formally,

While there exist ℓ, k such that $n_{\ell,k} > 0$,
  Assign any unassigned job j of length ℓ to any machine i s.t. $S_{i,k} = \emptyset$.
    Set $S_{i,k} = j$, $n_{\ell,k} = n_{\ell,k} - 1$.

The number of migrations is the number of non-dummy jobs assigned in step 4. This number is the minimal budget required to reach an optimal schedule. We prove the optimality of the algorithm by combining two lemmas.

Lemma 2.6 *The algorithm produces an optimal schedule with respect to the total flow-time.*

Proof: The schedule S satisfies the $n_{\ell,k}$ values calculated by SPT algorithm, therefore it must be optimal. Since these values were calculated according to the amounts of jobs in the modified instance, all jobs are assigned, that is, in Step 4, while there exist ℓ, k such that $n_{\ell,k} > 0$, it is guaranteed that there is an available empty slot for a job of length ℓ in round k. ■

Lemma 2.7 *Every schedule minimizing the total flow-time requires at least the same number of migrations as the number of migrations applied by the algorithm.*

Proof: We prove the following greedy choice property: *for every round k there exists an optimal solution minimizing the total number of migrations, in which the non-migrating jobs assigned to round k are identical to those selected by the algorithm.* The following simple observation will be used to analyze the assignment of jobs having lengths in L_2.

Observation 2.8 *For every round k, there are at most two lengths $\ell_1, \ell_2 \in L_2$ such that $n_{\ell_1,k} > 0$ and $n_{\ell_2,k} > 0$.*

Proof: By definition, jobs of lengths in L_2 span across more than one round in any optimal schedule. Another known property of SPT schedules is that all job lengths in round k are not shorter than job lengths in round $k-1$ and not longer than job lengths in round $k+1$. It is not possible to have three different lengths, all spanning over round k and an additional round, since in order to preserve the above SPT property, jobs of the middle length, must all be assigned to round k. ∎

We prove the greedy choice property for round k: Assume that an optimal schedule agrees with the algorithm in rounds earlier than k, and consider the assignment to round k. For every machine i, if $ForFree(i,k) = \emptyset$ then this is valid also for the optimal assignment, and a migration from another machine to $S_{i,k}$ is inevitable. If $ForFree(i,k)$ includes at least one job then we use exchange argument to show that any selection of job to $S_{i,k}$ that is different from the algorithm's choice can be changed to the algorithm's choice without hurting the total number of non-migrating jobs. Let $j \in ForFree(i,k)$ be the job assigned by the algorithm to $S_{i,k}$. Let $j' \neq j$ be the job assigned in the optimal schedule to $S_{i,k}$. If $j' \notin ForFree(i,k)$, then by switching j and j', we can only reduce the number of non-migrating jobs. If $j' \in ForFree(i,k)$, we distinguish between two cases:

1. $p_j \in L_1$. In this case, j must be assigned to round k, and assigning it to $S_{i,k}$ is the only way to assign it for free. By switching the assignment of j' and j in the optimal assignment, we avoid the migration of j, and cause a migration to j', thus, the total number of migrations does not increase.

2. $p_j \in L_2$. Since the algorithm gives priority to jobs whose lengths are in L_1, it must be that all job lengths in $ForFree(i,k)$ are in L_2 and in particular, $p_{j'} \in L_2$. By Observation 2.8, p_j, p'_j are the only lengths of jobs in $ForFree(i,k)$. Among lengths in L_2, the algorithm gives priority to shorter jobs, therefore, $p_j < p_{j'}$. Moreover, k is the last round in which jobs of length p_j will be assigned, as otherwise, the SPT order is not preserved (given that jobs of length $p_{j'}$ are assigned on both k and $k+1$). Therefore, assigning j to $S_{i,k}$ is the only way to assign it for free. By switching the assignment of j' and j in the optimal assignment, we avoid the migration of j, and cause a migration to j', thus, the total number of migrations does not increase.

We conclude that any optimal assignment can be modified such that it agrees with the algorithm's choice, without hurting the number of migrations. Thus, the algorithm produces an optimal assignment. ∎

Time complexity analysis: Algorithm 2 consists of four steps. In order to calculate the $n_{\ell,k}$ values in Step 1 the jobs should be sorted by processing times. If the initial schedule S_0 is arbitrary, or if the modification includes jobs addition or jobs' length modification, then the sorting takes in $O(n \log n)$ time. If the initial schedule is optimal, that is, in SPT order, and the modification does not include jobs' length modification, then the algorithm only needs to sort the jobs of each round in S_0 separately, and concatenate the resulting lists. As there are m_0 jobs in each round we get an $O(n \log m_0)$ time algorithm. If in the initial SPT schedule the jobs are assigned sequentially on the machines, or if m_0 is a constant, then Step 1 takes $O(n)$ time.

The partition of job lengths into L_1, L_2 in Step 2 is clearly linear. Step 3 iterates on the rounds and in each round assigns jobs using the already sorted list. The $ForFree$ structure can be implemented using a list of pointers. Since $ForFree$ jobs are assigned in a non-decreasing order and since, by observation 2.8, at most two different lengthes from L_2 are considered in each round, Step 3 takes $O(m)$ for each round and a total of $O(m\frac{n}{m}) = O(n)$. In step 4, the algorithm assigns the remaining jobs in time $O(n)$.

We conclude that the time complexity of the algorithm varies between $O(n)$ and $O(n \log n)$, depending on the initial schedule and the allowed modification in the instance.

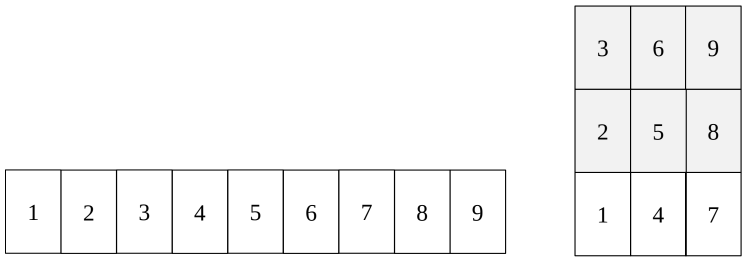
2.4 Linear-time algorithms for some special cases

Algorithm 2 uses the minimal budget required to achieve an optimal schedule assuming unit migration cost. For several cases of m_0, m, and when the original schedule is optimal (SPT) we can calculate an optimal solution in linear time and determine the required budget in constant time. We consider the two cases of adding or removing machines, where in each case we explore several options and show the analysis of the minimal budget bound. Recall that M_0 denotes the initial set of machines and $m_0 = |M_0|$, M denotes the modified set of machines, and $m = |M|$. Let M' denote the set of added / removed machines, and $m' = |M'|$. Thus, in machhines' addition, $m' = m - m_0$ and in machines' removal, $m' = m_0 - m$. Denote by S_0, S the original and the modified schedule respectively. Let R_{0k}, R_k denote the k^{th} round in the initial and in the modified schedule, respectively.

2.4.1 Adding machines

The case $m_0 \leq m'$: we show that the optimal schedule for this case can be achieved using only migrations to the new machines (there are no internal migrations). Specifically, we show that there exists an optimal schedule in which no job scheduled on M_0 migrates to a different machine in M_0. Thus, the minimal budget is the minimum number of jobs on the new machines in an optimal schedule, which is $m' \left\lfloor \frac{n}{m_0+m'} \right\rfloor$. As S_0 is optimal (SPT), there exists an optimal schedule S where for all k, each of the jobs of R_{0k} is scheduled in S in round that is not higher than any round on which a job from R_{0k+1} is assigned.

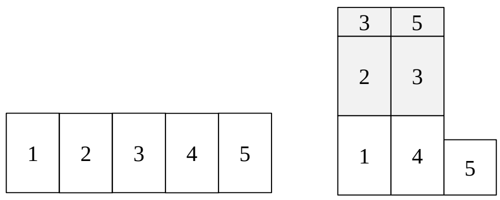
If m' is a multiple of m_0, that is, for some integer x, $m' = xm_0$, there exists an optimal schedule where every $x + 1$ rounds of S_0 unite to one round of S. Therefore, in every round R_k it is possible to assign m jobs on their original machines (see Figure 5 for $x = 2$). Specifically, assigning jobs from the first round of the $x + 1$ rounds on their original machines in S. Clearly, such an assignment is optimal and as all machines $m \in M_0$ are assigned with non migrating jobs, there is no optimal schedule with less migrations.



(a) (b)

Figure 5: (a) The initial schedule (b) An optimal modified schedule when $m' = 2m_0$.

If $m_0 \geq m'$, and m' is not a multiple of m_0, it might be that jobs from one round in S_0 will end up in different rounds in S. Still, we show that there exists an optimal schedule S achieved using no internal migrations. Thus, a budget of $m' \left\lfloor \frac{n}{m_0+m'} \right\rfloor$ is sufficient. Since $m_0 < m'$, the number of jobs in every round in S is more than $2m_0$, therefore, every round of S includes at least one whole round of S_0. This implies that in every round of S it is possible to assign m jobs on their original machines. Clearly, such assignment is optimal and as all machines $m \in M_0$ are assigned with non migrating jobs, there is no optimal schedule with less migrations. Such an assignment is demonstrated in Figure 6. The second round includes jobs from R_{03}, R_{04}, R_{05} where R_{04} is fully included in R_2. It is possible to assign all the jobs in R_{04} on their original machines.



(a) (b)

Figure 6: (a) The initial schedule (b) An optimal modified schedule when $m_0 < m'$

The case $m_0 > m'$: In this case, as demonstrated in Figure 2 in the introduction, it might be inevitable to have internal migrations within M_0. We cannot bound the minimal required budget by $m' \left\lfloor \frac{n}{m_0+m'} \right\rfloor$, and Algorithm 2 should be applied.

Finally, we note that as demonstrated in Figure 7, if the initial schedule S_0 is not optimal or if the modification includes changes of job lengths, the bound is not valid even in the simplest case $m' = 1$.

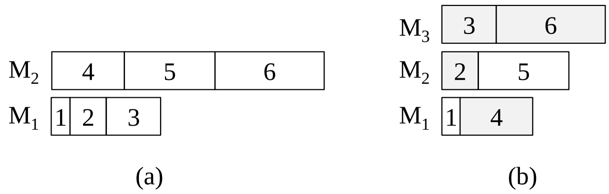


Figure 7: (a) An initial non optimal assignment, (b) M_3 is added, an optimal reassignment requires 4 migrations.

2.4.2 Removing machines

When m' is a multiple of m: Assume that for some integer x, $m' = xm$. We show that the minimal budget required for achieving an optimal schedule in this case is the number of jobs on the removed machines. Specifically, we show that there exists an optimal schedule in which no job from $M_0 \setminus M'$ migrates.

As $m' = xm$, there exists an optimal schedule where every round of S_0 forms $x+1$ subsequent rounds of S. For all k, the jobs of R_{0k} that are assigned on machines that are not removed might spread on $x+1$ different rounds. Still, each of these jobs can remain on its machine (see Figure 8 for $x = 2$). Clearly, such an assignment is optimal and, as all machines $m \in M_0$ are assigned with non migrating jobs, there is no optimal schedule with less migrations.

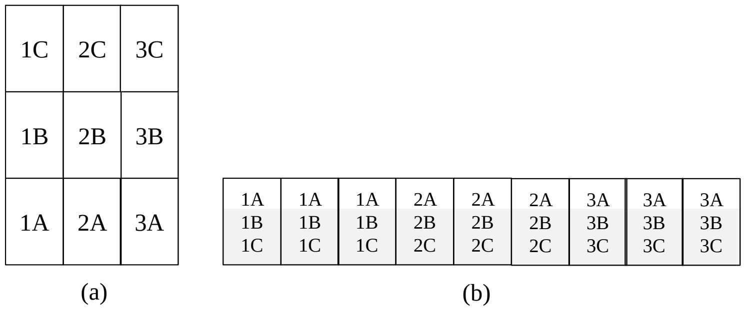


Figure 8: (a) The initial schedule (b) An optimal modified schedule when $m' = 2m$.

When $(m' \bmod m) \neq 0$: In this case internal migrations within M might be inevitable. For $m > m'$, consider the example in Figure 9. The initial schedule on $m_0 = 4$ machines is optimal. Assume that M_4 is removed. Any optional modified schedule must satisfy $n_{34} = 3$, thus one internal migration is inevitable - a job of length 4 must leave M_3.

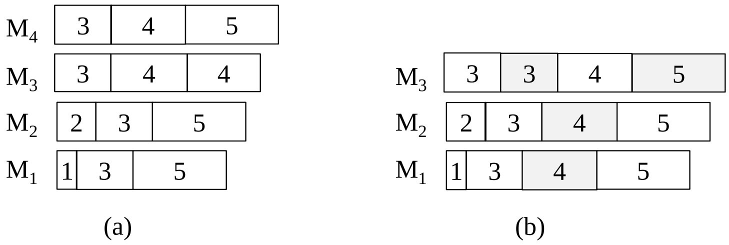


Figure 9: (a) An initial optimal assignment, (b) M_4 is removed, an optimal reassignment requires 4 migrations.

For $m \leq m'$, consider consider the example in Figure 10(a) and 10(b). The initial schedule on $m_0 = 5$ machines is optimal. Assume that 3 machines, M_3, M_4, M_5 are removed. Any optional modified schedule must satisfy $n_{32} = 1$ and $n_{22} = 1$, thus one internal migration is inevitable - one job must leave M_1.

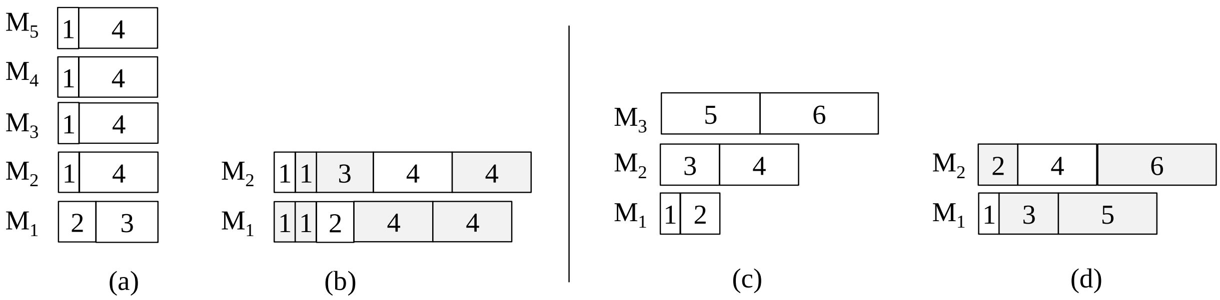


Figure 10: (a) An initial optimal assignment, (b) M_3, M_4, M_5 are removed, an optimal reassignment requires 7 migrations. (C) An initial non-optimal assignment, (d) M_3 is removed, an optimal reassignment requires 4

Thus, when $(m' \bmod m) \neq 0$ for both $m < m'$ and $m > m'$, internal migrations might be inevitable and a budget of $(m_0 - m') \left\lfloor \frac{n}{m_0} \right\rfloor$ might not be sufficient.

Finally, we note that as demonstrated in Figure 10(c) and 10(d), if the initial schedule S_0 is not optimal or if the modification includes changes of job lengths, the bound is not valid even in the simplest case $m' = 1$.

3 Rescheduling with a Limited Budget - Unit Migration Costs

In this section we consider the rescheduling problem assuming a limited budget. Naturally, the goal is to utilize the budget in the best possible way, that is, the modified schedule should have a low total flow-time – the minimal possible among all schedules that can be achieved using the given budget. We assume unit migration costs, that is, $\theta_{ii'j} = 1$, independent of the job j and the involved machines. Thus, the budget B gives the maximal number of allowed migrations. We also assume that $n > B$, as otherwise an optimal schedule can be found by ignoring the migration costs.

The problem can be described as the following weighted matching problem: Similar to the technique used in Section 2.1, let $G = (V, E)$, be a complete bipartite graph with n nodes on one side and mn nodes in the other side. The node q_{ik}, for $i = 1, \ldots, m$ and $k = 1, \ldots, n$, corresponds to the k^{th} from last position on machine i. The edge (j, q_{ik}) has weight kp_j, reflecting the contribution of j to the total flow-time if it is assigned on the k^{th} from last position on machine i. We color the edges of G as follows: If an edge (j, q_{ik}) corresponds to a migration of j, that is, i is not the machine j is assigned to in S_0, then the edge is red, otherwise the edge is blue.

It is easy to verify that a min-weight perfect matching with at most B red edges corresponds to an optimal reschedule. For an arbitrary bipartite graph with arbitrary weights, the complexity of the above restricted matching problem is unknown. Some special cases for which efficient algorithms exist include bipartite graphs with unit-weights [18], or with equal sizes ($K_{n,n}$) [31]. The more general problem of determining whether a complete weighted bipartite graph has a complete matching with a specific weight w in known to be NP-hard [7]. We present optimal polynomial time algorithms for several classes of instances of our problem.

3.1 The budget B is a constant

Assume that the modification occurs at time $t = 0$, and the budget B is a constant. Clearly, every job j may either migrate or not, and as the budget is a constant, there are at most $\binom{n}{B} < n^B$ possible ways to select the subset of jobs that are allowed to migrate. The following algorithm considers each selection separately.

Algorithm 3 An optimal algorithm for rescheduling when the budget B is a constant

---

For every possible selection of B jobs $J' \subset J$:

1. Let $G = (V, E)$, be a bipartite graph with n nodes on one side and mn nodes in the other side. The node q_{ik}, for $i = 1, \ldots, m$ and $k = 1, \ldots, n$, corresponds to the k^{th} from last position on machine i. For every job $j \in J'$, there is an edge (j, q_{ik}) for every $i = 1, \ldots, m$ and $k = 1, \ldots, n$. For every job $j \notin J'$, there is an edge (j, q_{ik}) for every $k = 1, \ldots, n$, but only for the machine i on which j is assigned to in S_0. The weight of (j, q_{ik}) is kp_j.
2. Find a min-weight complete matching in G.

Return the schedule induced by the min-weight matching.

Theorem 3.1 *Algorithm 3 returns a modified schedule whose total flow-time is minimal among all schedules achieved with budget at most B.*

Proof: Let S^* be an optimal modified schedule. Let J^* denote the set of migrating jobs in S^*. Consider the iteration in which $J' = J^*$. The bipartite graph built in this iteration includes edges connecting vertices corresponding to jobs from J^* to vertices corresponding to slots on all the machines, in particular, to machines that are different from their original machines. The weight of an edge (j, q_{ik}) corresponds to the contribution of j to the total flow-time if it is assigned on the kth from last position on M_i. Therefore the min-weight matching found in this iteration induces the schedule S^*. ∎

3.2 Migrations are allowed only to new machines

Another case for which it is possible to solve the problem optimally is when the system's modification consists of machines addition and the only allowed migrations are to the new machines. This scenario arises in practice when the system is upgraded with new machines that are ready to receive tasks, while the old machines are not capable to accept new tasks. We present an optimal algorithm for this problem based on a reduction to a min-cost max-flow problem. An illustration of the flow network is given in Figure 11. Each edge is labeled by its capacity and the cost of one flow unit.

An overview of the flow network: The set of nodes r_{ik} for $1 \leq i \leq m_0,\ 1 \leq k \leq n$ correspond to positions on the initial machines. The set of nodes $q_{i'k}$ for $1 \leq i' \leq m',\ 1 \leq k \leq B$ correspond to positions on the added machines. All the q-nodes are connected to node d. The capacity of the edge (d,t) is the budget B. This limited capacity guarantees that the total number of slots occupied on the new machines will not exceed B. The set of nodes $1 \leq j \leq n$ correspond to the jobs. Every job j that is assigned to machine i in S_0 is connected to the nodes corresponding to positions on machine i and to all the q-nodes. The capacities of all edges except for (d,t) are 1. The cost of an edge connecting job j to a node corresponding to a k^{th} from last position (on any machine) is kp_j. All other edges have cost 0.

Theorem 3.2 *A minimum-cost maximum-flow (of value n) in G corresponds to an optimal schedule without exceeding the budget B.*

Proof: First, note that every valid schedule corresponds to a maximum-flow in G. On the other hand, not every maximum-flow in G corresponds to a schedule, since a job might be assigned to the k^{th} from last position in some machine, while less than k jobs are assigned to that machine. However, such a maximum-flow is clearly not of minimal cost - a better matching can be obtained by shifting the $k' < k$ jobs assigned to that machine into the k' last slots. Therefore, a schedule of minimum total flow-time corresponds to a minimum-cost maximum-flow in G.

As the capacity of (d,t) is B, while all other edges' capacity is 1, at most B q-nodes have incoming flow. These nodes correspond to migrating jobs. Thus, a minimum-cost maximum-flow in G corresponds to an optimal schedule without exceeding the budget B. ∎

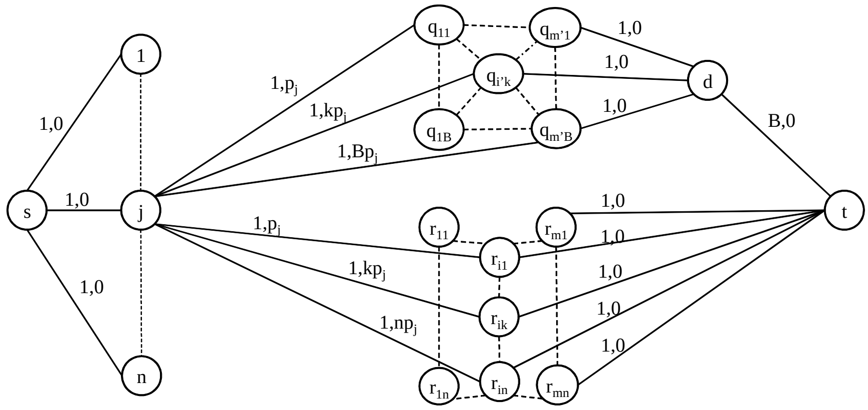


Figure 11: The flow network built for the rescheduling with limited budget problem.

This algorithm can be extended for the case in which the systems' modification occurs at time $t > 0$ - similar to the extensions described in Section 2.2. If restarts are allowed, then our extension assumes that every currently processed job is the shortest job on its machine (which is true if the initial schedule is optimal, or if the schedule is a result of our algorithm - even on a sequence of modifications). If restarts are not allowed then our extension is valid for any initial schedule.

4 Conclusions and Future Work

We studied reoptimization problems arising in production planning, in which the goal is to combine the objective of finding a schedule with low total flow-time, with the goal of efficiently converting a given initial schedule to the modified one. We presented the first positive results in this framework. We presented algorithms for finding an optimal schedule achieved using the minimal possible transition cost, and algorithms for optimal utilization of a limited number of migrations.

Several interesting important problems remain open:

1. Identify the complexity status of the second problem for arbitrary transition costs and arbitrary modifications. As explained in Section 3, even with unit transition costs this is a special case of a more general open problem (min-weight matching with limited number of red edges).

2. Identify the range of budget B for which it is guaranteed that an optimal reschedule can be achieved using no internal migrations. It is easy to see that this range is included in $m' < B \leq m' \frac{n}{m_0+m'}$.

3. Another open research direction is to consider different objective functions. In particular, minimizing the makespan of the schedule, given by the last completion time of some job. Since the problem is NP-hard, the reoptimization problem is clearly also NP-hard. The goal is to develop an algorithm for the reoptimization problem whose approximation-ratio is similar to the best approximation-ratio known for the original problem. For the minimum total-flow problem, we were able to reduce the reoptimization problem on identical parallel machines to the same problem on unrelated machines ($R||\sum C_j$). It seems that a similar reduction can be applied also for the minimum makespan problem. The best approximation ratio for the resulting problem ($R||C_{max}$) is $2-1/m$, and it is based on solving an LP problem [29]. We believe that a simpler greedy algorithm tailored for the reoptimization problem can have a similar performance. Note that the order of the jobs assigned to a specific machine is not important. Thus, some of the challenges involved in scheduling remainders of currently processed jobs as first on their machines are not relevant in this problem.

References

[1] G. Amato, G. Cattaneo, and G. F. Italiano. Experimental analysis of dynamic minimum spanning tree algorithms. *In Proc. of* 8th SODA, 1997.

[2] C. Archetti, L. Bertazzi, and M.G. Speranza. Reoptimizing the 0-1 knapsack problem. *Discrete applied mathematics*, vol. 158(17), 2010.

[3] G. Ausiello, V. Bonifaci, and B. Escoffier. Complexity and approximation in reoptimization. In *Computability in Context: Computation and Logic in the Real World*, B. Cooper, A. Sorbi Eds., Imperial College Press/World Scientific, 2011.

[4] G. Ausiello, B. Escoffier, J. Monnot, and V. Th. Paschos. Reoptimization of minimum and maximum traveling salesmans tours. *J. of Discrete Algorithms* 7(4):453-463, 2009.

[5] H.J. Böckenhauer, L. Forlizzi, J. Hromkovič, J. Kneis, J. Kupke, G. Proietti, and P. Widmayer. On the approximability of TSP on local modifications of optimally solved instances. *Algorithmic Operations Research* 2(2), 2007.

[6] J.L. Bruno, E.G Coffman, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, 17:382-387, 1974.

[7] R. Chandrasekaran, S.N. Kaboadi, and K.G. Murty. Some NP-complete problems in linear programming. *Operations Research Letters*, 1:101-104, 1982.

[8] R.W. Conway, W.L. Maxwell, and L.W. Miller. Theory of Scheduling. *AddisonWesley*, 1967.

[9] C. Demetrescu, I. Finocchi, and G.F. Italiano. Dynamic graph algorithms. *Handbook of Graph Theory*, J. Yellen and J.L. Gross eds., CRC Press Series, in Discrete Math and Its Applications, 2003.

[10] J. Edmonds, and R. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM*, 248-264 (1972).

[11] D. Eppstein, Z. Galil, and G.F. Italiano. Dynamic graph algorithms, Chapter 8. In *CRC Handbook of Algorithms and Theory of Computation*, ed. M. J. Atallah, 1999.

[12] B. Escoffier, M. Milanič, and V. Th. Paschos. Simple and fast reoptimizations for the Steiner tree problem. *DIMACS Technical Report* 2007-01.

[13] L.R. Ford, and D.R. Fulkerson. Flows in networks. *Princeton Univ. Press, Princeton*, NJ (1962).

[14] A.V. Goldberg, and R.E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. Assoc. Comput. Mach.*, 36(4):873-886, 1989.

[15] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Math.* 5, 287-326, 1979.

[16] F. Grandoni, R. Zenklusen. Optimization with more than one budget. In *Proc. of ESA*, 2010.

[17] W. Horn. Minimizing average flow-time with parallel machines. *Operations Research*, 21:846-847, 1973.

[18] A.V. Karzanov. Maximum matching of given weight in complete and complete bipartite graphs. *Kibernetika*, 1:7-11, 1987. English translation in CYBNAW 23(pp. 8-13).

[19] J. R. Kenney Solving unweighted and weighted bipartite matching problems in theory and practice. Ph.D Thesis, Stanford University, 1995.

[20] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955.

[21] D. Mattox. Handbook of Physical Vapor Deposition (PVD) Processing, 2nd Edition. *Elsevier*, 2010.

[22] E. Nardelli, G. Proietti, and P. Widmayer. Swapping a failing edge of a single source shortest paths tree is good and fast. *Algorithmica* 35, 2003.

[23] S. Pallottino, and M. G. Scutella. A new algorithm for reoptimizing shortest paths when the arc costs change. *Operations Research Letters*, vol. 31, 2003.

[24] R. Ravi, and M. X. Goemans. The constrained minimum spanning tree problem. In *5th Workshop on Algorithm Theory*, 66-75, 1996.

[25] H. Shachnai, G. Tamir, and T. Tamir, Minimal cost reconfiguration of data placement in storage area network. *Theoretical Computer Science*. vol. 460, pages 42-53, 2012.

[26] H. Shachnai, G. Tamir, and T. Tamir. A theory and algorithms for combinatorial reoptimization. *In Proc of* 10th LATIN, April 2012.

[27] R.A Sitters. Two NP-hardness results for preemptive minsum scheduling of unrelated parallel machines. Proc. 8th International IPCO Conference, LNCS, Springer, 396-405, 2001.

[28] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*. vol. 3, pp. 5966, 1956.

[29] E.V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33:127–133, 2005.

[30] M. Thorup, and D.R. Karger. Dynamic graph algorithms with applications. *In Proc of* 7th SWAT, 2000.

[31] T. Yi, K.G. Murty, and C. Spera. Matchings in colored bipartite networks. *Discrete Applied Mathematics*, 121:261-277, 2002.