Tight running times for minimum ℓ_q-norm load balancing: beyond exponential dependencies on $1/\epsilon$

Lin Chen* Liangde Tao† José Verschae‡

Abstract

We consider a classical scheduling problem on m identical machines. For an arbitrary constant $q > 1$, the aim is to assign jobs to machines such that $\sum_{i=1}^{m} C_i^q$ is minimized, where C_i is the total processing time of jobs assigned to machine i. It is well known that this problem is strongly NP-hard.

Under mild assumptions, the running time of an $(1+\epsilon)$-approximation algorithm for a strongly NP-hard problem cannot be polynomial on $1/\epsilon$, unless P = NP. For most problems in the literature, this translates into algorithms with running time at least as large as $2^{\Omega(1/\varepsilon)} + n^{O(1)}$. For the natural scheduling problem above, we establish the existence of an algorithm which violates this threshold. More precisely, we design a PTAS that runs in $2^{\tilde{O}(\sqrt{1/\epsilon})} + n^{O(1)}$ time. This result is in sharp contrast to the closely related minimum makespan variant, where an exponential lower bound is known under the exponential time hypothesis (ETH). We complement our result with an essentially matching lower bound on the running time, showing that our algorithm is best-possible under ETH. The lower bound proof exploits new number-theoretical constructions for variants of progression-free sets, which might be of independent interest.

Furthermore, we provide a fine-grained characterization on the running time of a PTAS for this problem depending on the relation between ϵ and the number of machines m. More precisely, our lower bound only holds when $m = \Theta(\sqrt{1/\epsilon})$. Better algorithms, that go beyond the lower bound, exist for other values of m. In particular, there even exists an algorithm with running time polynomial in $1/\epsilon$ if we restrict ourselves to instances with $m = \Omega(1/\epsilon \log^2 1/\epsilon)$.

1 Introduction

We consider a classical scheduling problem on identical parallel machines. Suppose we are given m identical machines and n jobs, each having a processing time p_j. A feasible solution corresponds to an assignment of jobs to machines. For a given assignment, let C_i be the total processing time of jobs assigned to machine i, that is, $C_i = \sum_{j \to i} p_j$. Our objective is to minimize $\sum_{i=1}^{m} C_i^q$, where $q > 1$ is an arbitrary constant. For either exact algorithms or approximation schemes, minimizing $\sum_{i=1}^{m} C_i^q$ is equivalent to minimizing the ℓ_q-norm of machine loads, i.e., $(\sum_{i=1}^{m} C_i^q)^{1/q}$. In the standard 3-field scheduling notation by Graham et al. [13], this problem is denoted as $P||\sum_i C_i^q$.

Our problem is well-known to be strongly NP-hard by a simple reduction from 3-partition. On the other hand, a classic result by Alon et al. [1] shows that it admits a polynomial time approximation scheme (PTAS) with running time $f(1/\epsilon) + n^{O(1)}$, where $f(1/\epsilon)$ is doubly exponential in $1/\epsilon$. Very recently, improved running times have been obtained for $P||\sum_i C_i^q$ and other closely related load balancing problems. Particularly, for a variety of objective functions, which include both $\sum_i C_i^q$ and the makespan objective $C_{\max} = \max_i C_i$, Jansen et al. [20] show that the problem admits a PTAS with a running time of $2^{\tilde{O}(1/\epsilon)} + \tilde{O}(n)$. On the negative side, for the makespan objective, Chen et al. [7] show that such a running time is essentially best possible under the exponential time hypothesis (ETH). However, the lower bound does not hold for other objectives, including $P||\sum_i C_i^q$, leaving open the possibility for improved running times. In this paper, we study this question and explore the surprisingly rich complexity landscape of $P||\sum_i C_i^q$ in the context of approximation schemes.

Contribution Overview. We study the complexity landscape of approximation schemes for $P||\sum_i C_i^q$. Consider some strongly NP-hard optimization problem whose optimal value $\mathrm{OPT}(I)$ is integral and upper bounded by $\mathrm{poly}(|I|_u)$ for any instance I, where $|I|_u$ is the input size written in unary. This implies that the problem does not admit a fully polynomial-time approximation scheme (FPTAS) unless P=NP [11]. In the majority of cases,

*chenlin198662@gmail.com, Department of Computer Science, Texas Tech University, US.
†vast.tld@gmail.com, Department of Computer Science, Zhejiang University, China.
‡jverschae@uc.cl, Institute for Mathematical and Computational Eng., Pontificia Universidad Católica de Chile, Chile.

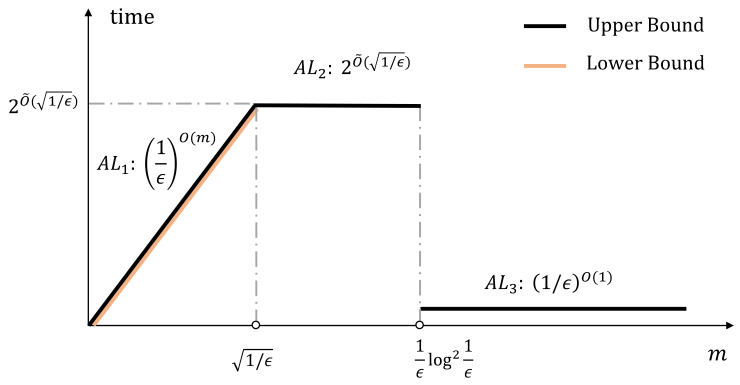

Figure 1: Complexity landscape of $P||\sum_i C_i^q$. The time axis specifies the dependency of the running time with respect to $1/\epsilon$. A term of $n^{O(1)}$ needs to be added in the running of each algorithm.

for such problems the literature presents PTASs with running time at least as large as $2^{\Omega(1/\epsilon)} + n^{O(1)}$, that is, the dependency on $1/\epsilon$ is exponential. We show that $P||\sum_i C_i^q$ does not fall into this case, and a running time subexponential on $1/\epsilon$ is achievable. More precisely, we give a PTAS with a running time of $2^{\tilde{O}(\sqrt{1/\epsilon})} + n^{O(1)}$. On the other hand, we show that this running time is essentially tight, by providing an almost matching lower bound under ETH. That is, we show that ETH rules out a PTAS of running time $2^{O((1/\epsilon)^{1/2-\delta})} + n^{O(1)}$ for any $\delta > 0$. We are not aware of any other PTAS for a strongly NP-hard problem with such a tight subexponential behavior on $1/\varepsilon$.

Besides the results above, we give a fine-grained study on the upper and lower bounds of the running time of a PTAS for $P||\sum_i C_i^q$. First of all, we notice that our lower bound only holds for a small range of values of m, depending on ϵ. Moreover, for some other values, we can circumvent the lower bound and obtain improved running times. More precisely, the lower bound only holds when $m = \Theta(\sqrt{1/\epsilon})$. Quite surprisingly, when m is larger, namely $m = \Omega(1/\epsilon \log^2(1/\epsilon))$, an algorithm that runs polynomially in $1/\epsilon$ exists, despite the problem being strongly NP-hard in general and our stronger lower bound. If $m = O(\sqrt{1/\epsilon})$ we can use a PTAS with running time $(1/\epsilon)^{O(m)}$, which also breaks the lower bound for $m = o(\sqrt{1/\epsilon})$. See Figure 1 for a depiction of our results. It remains an open problem to obtain tight running times when $m = \Theta((1/\epsilon)^\theta)$ for $\theta \in (1/2, 1]$.

Technical Contribution. Our main technical contribution lies in the lower bound proof. For this, we give a fine-grained reduction from a variant of Max3SAT to $P||\sum_i C_i^q$. To do so, we convert a set of clauses to a set of jobs. We enforce that two jobs which represent variables in the same clause are scheduled together in some carefully constructed gap (i.e., slot) of a given size. For such a construction, it is imperative to use pairs of numbers with unique sums, to guarantee that only these two jobs fit this gap. Hence, our construction is tightly related to *Sidon sets* and *Salem–Spencer sets* (also called progression-free sets), both of which have been studied extensively in number theory (see, e.g., [10, 35, 33, 12]). A Sidon set $S = \{s_1, s_2, \ldots, s_n\}$ is a subset of natural numbers where all pairwise sums $s_i + s_j$, for $i \leq j$, are distinct. That is, $s_i + s_j = s_{i'} + s_{j'}$ implies $\{i, j\} = \{i', j'\}$. A weaker notion is that of a Salem-Spencer set, that is, a set $S = \{s_1, s_2, \cdots, s_n\}$ with no cardinality 3 progression, i.e., no triplet $(i, j, k) \in \mathbb{Z}_n := \{1, 2, \cdots, n\}$ of pairwise different numbers satisfies $s_i - s_j = s_k - s_i$. In other words, if $s_j + s_k = 2s_i$ then $i = j = k$. Our lower bound could be proved by adapting known techniques if a Sidon set $S \subseteq \mathbb{Z}_N$ (where $\mathbb{Z}_N := \{1, 2, \cdots, N\}$) with cardinality n exists for $N = n^{1+o(1)}$. Unfortunately, this is impossible, as Erdös and Turán [10] show that the cardinality of a Sidon set with n elements requires $N = \Omega(n^2)$. We can circumvent this negative result by requiring only some pairs of numbers to have a unique sum, where these pairs correspond to the clauses in the given Max3SAT instance. Towards this, we first transform the given Max3SAT instance, with variables z_j for $j \in \mathbb{Z}_n$, into a special structure such that all clauses can be divided into two disjoint subsets C_1 and C_2: C_1 consists of clauses $cl_2, cl_5, \cdots, cl_{n-1}$ such that $cl_\ell = (w_{\ell-1} \vee w_\ell \vee w_{\ell+1})$, where $w_j \in \{z_j, \neg z_j\}$ for all j; and C_2 consists of clauses $cl'_1, cl'_2, \cdots, cl'_n$ such that $cl'_\ell = (z_\ell \oplus \neg z_{\tau(\ell)})$, where τ is a permutation of $\mathbb{Z}_n$ and $\oplus$ is the XOR operation (see Section 3.1 for details). For C_1, we construct a set of numbers $\{\sigma(1), \sigma(2), \cdots, \sigma(n)\}$ such that every adjacent sum $\sigma(i) + \sigma(i+1)$ is unique, and this will be

achieved through extending a known construction of Salem-Spencer sets (Lemma 3.3). For C_2, we extend the construction to additionally require that the $\sigma(i)$'s we construct admit a *linked unique sum*. That is, there exists a subset of numbers $E_i = \{e_{i,1}, e_{i,2}, \ldots, e_{i,\omega}\} \subseteq \mathbb{Z}_{n^{1+o(1)}}$ for every $i \in \mathbb{Z}_n$ such that $E_i \cap E_{i'} = \emptyset$ for any $i' \neq i$, and the sum of each pair $\sigma(i) + e_{i,1}, e_{i,1} + e_{i,2}, \cdots, e_{i,\omega-1} + e_{i,\omega}, e_{i,\omega} + \sigma(\tau(i))$ is unique in the sense that no other pairs in $S \cup E$ sum up to the same value, where $E = \bigcup_{i=1}^{n} E_i$. Note that a linked unique sum is a weaker notion than Sidon or Salem-Spencer, as for these there is no auxiliary set E. Nevertheless, the property of linked unique sum is strong enough for our reduction. The construction of the auxiliary set E relies on further extending our technique for constructing unique adjacent sums, together with a group theoretic lemma that allows an "orthogonal" decomposition of the permutation τ (Lemma 3.5). Our results may be of separate interest for constructing fine-grained lower bounds on approximation or parameterized algorithms for other problems.

Another crucial observation, which may also be of independent interest, is a structural result needed for our PTAS with running time $2^{\tilde{O}(\sqrt{1/\epsilon})} + n^{O(1)}$ (see Section 2.1). For many objective functions (like $C_{\max}$) we can round the processing times to powers of $1+\epsilon$ in order to bound the overall loss by a factor of $1 + O(\epsilon)$. We observe that for minimizing $\sum_i C_i^q$ it is possible to consider a coarser grouping of jobs into sizes within a $(1+\sqrt{\epsilon})$ factor. Broadly speaking, by imposing extra structure to a near-optimal solution, we can use a Taylor expansion to bound the error, and notice that the linear term of the polynomial expansion cancels out. This leaves us only with the quadratic (and lower order) terms. This observation might translate to other problems with ℓ_q-norm objective, and even other min-sum cost functions.

Related Work. Load balancing problems are fundamental in computer science and have been studied extensively in the literature. In particular, the first PTAS for $P||C_{\max}$ dates back to the 80's [15] and there is a long history of improvements on the running time for various identical machine scheduling problems, including $P||C_{\max}$, $P||\sum_i C_i^q$, $P||\sum_j w_j C_j$, etc.; see, e.g., [27, 2, 14, 37, 19, 20]. Recently, more general objective functions based on arbitrary norms have been considered [16]. Parameterized algorithms for scheduling problems have also been studied extensively (see, e.g. [22, 31, 32, 25, 8]).

The exponential time hypothesis (ETH) is a widely accepted complexity assumption introduced by Impagliazzo et al. [18, 17], which can be used to obtain lower bounds on the running time of algorithms for various problems (see, e.g., [28] for a survey). In 2014, Chen et al. [7] provide a concrete lower bound on the running time of a PTAS for $P||C_{\max}$ under ETH. Later, Jansen et al. [20] give a PTAS with running time $2^{\tilde{O}(1/\epsilon)} + n^{O(1)}$ for $P||C_{\max}$, which almost matches the lower bound.

Despite PTASs having been established for a variety of optimization problems, much less is known regarding lower bounds on their running time. In addition to $P||C_{\max}$, mentioned above, other well-known examples include multiple knapsack [21], planar vertex cover, planar dominating set, and planar traveling salesperson [29]. Interestingly, all of these lower bounds have an almost linear dependency on $1/\epsilon$ in the exponent, which essentially matches the best-known PTAS. Generally, Chen et al. [6] proved that if the problem, parameterized by $1/\epsilon$, is W[1]-hard under a linear FPT reduction, then there is no PTAS with $f(1/\epsilon)|I|^{o(1/\epsilon)}$ running time for an arbitrary computable function f, assuming all problems in SNP cannot be solved in sub-exponential time. We are not aware of a PTAS whose running time is subexponential in $1/\epsilon$, either for scheduling or other strongly NP-hard problems.

Unlike approximation algorithms, subexponential running times on a parameter have been observed in the field of parameterized algorithms and have received significant attention. In particular, a variety of optimization problems in planar graphs admit a fixed parameter tractable (FPT) algorithm that is subexponential in the parameter, including, e.g., independent set [9], dominating set [9], and multiway cut [24, 30, 36]. Note that, on the other hand, a subexponential PTAS was ruled out for the planar dominating set problem [29].

2 Approximation schemes

The goal of this section is to prove the following theorem.

THEOREM 2.1. *For any sufficiently small $\epsilon > 0$, there exists an algorithm that outputs a $(1+\epsilon)$-approximate solution for the scheduling problem $P||\sum_i C_i^q$ within $2^{\tilde{O}(\sqrt{1/\epsilon})} + n^{O(1)}$ time. More specifically, there exists a:*

- *$(1+\epsilon)$-approximation algorithm AL_1 that runs in time $(1/\epsilon)^{O(m)} + n^{O(1)}$ for $m = O(\sqrt{1/\epsilon})$;*
- *$(1+\epsilon)$-approximation algorithm AL_2 that runs in time $2^{\tilde{O}(\sqrt{1/\epsilon})} + n^{O(1)}$ for $m = (1/\epsilon)^{O(1)}$;*

- $(1+\epsilon)$-*approximation algorithm* AL_3 *that runs in time* $(1/\epsilon)^{O(1)} + n^{O(1)}$ *for* $m = \Omega(1/\epsilon \log^2(1/\epsilon))$.

In particular, for a sufficiently small ϵ, we may run AL_1 for $m \leq \sqrt{1/\epsilon}$, run AL_2 for $\sqrt{1/\epsilon} < m \leq 1/\epsilon^2$, and run AL_3 for $m \geq 1/\epsilon^2$. This guarantees a $2^{\tilde{O}(\sqrt{1/\epsilon})} + n^{O(1)}$ time algorithm for all values of m and $1/\epsilon$.

We remark that standard techniques round the processing time of a job to some multiple of $1+\epsilon$, yielding an instance with $\tilde{O}(1/\epsilon)$ different types of jobs. However, such rounded instance cannot be solved to optimality in time $2^{(1/\epsilon)^{1-\delta}} + n^{O(1)}$ for any constant $\delta > 0$ [7]. Hence, we need a new approach for Theorem 2.1.

We now give a brief overview of the proof of Theorem 2.1. Algorithm AL_1 is based on a standard dynamic programming, given in Appendix A.1. Algorithm AL_2 is based on an new observation (Lemma 2.3) which shows that we can classify processing times on intervals of the form $[\epsilon(1+\sqrt{\epsilon})^{h-1}, \epsilon(1+\sqrt{\epsilon})^h)$ for an integer h. After preprocessing the instance (Lemma 2.1), we can focus on only $\tilde{O}(1/\sqrt{\epsilon})$ such intervals. We show that there exists a near-optimal solution where jobs are scheduled in an ordered way following the mentioned classification. This algorithm is described in Section 2.1. Algorithm AL_3 (see Appendix A.3) is based on modifying the famous algorithm for the bin packing problem by Karmarkar and Karp [23].

All the three algorithms will operate on a scheduling instance that is well-structured, as implied by the following lemma. The structure can be achieved through standard techniques, namely scaling and grouping of small jobs, see, e.g., [2]. For an instance I, we denote by $\mathrm{size}(I)$ the total processing time of jobs in I, and by $m(I)$ the number of machines.

LEMMA 2.1. (ALON ET AL. [2]) *For any sufficiently small* $\epsilon > 0$*, given an arbitrary instance* I_0 *of* $P||\sum_i C_i^q$*, we can transform in linear time* I_0 *into a well-structured rounded instance* I *with less or equal number of jobs and less or equal number of machines, that satisfies:*

- $size(I) = m(I)$*;*
- *the processing time of each job in* I *belongs to* $[\epsilon, 1]$*;*
- *there exists an optimal solution for* I *such that the load of each machine belongs to* $[1/2, 2]$.

Furthermore, any $(1+\epsilon)$*-approximation solution for* I *can be transformed into an* $(1+O(\epsilon))$*-approximation solution for* I_0 *in linear time.*

In the following, we focus exclusively on the instance after the preprocessing. It is worth mentioning that for non-integral values of q, the objective function can be irrational even for rational processing times. For obtaining a PTAS this is however not a problem, as computing the objective function up to an additive error of $\epsilon/\mathrm{poly}(n)$ suffices for our results. In what follows we omit this technicality, and assume that we can compute the objective function without error.

2.1 Algorithm 2 In this subsection, we describe and analyze algorithm AL_2.

LEMMA 2.2. *Consider an instance after the preprocessing of Lemma 2.1. For any* $\epsilon > 0$*, there exists an algorithm* AL_2 *that outputs a* $(1+O(\epsilon))$*-approximation solution for* $P||\sum_i C_i^q$ *with* $m^{\tilde{O}(1/\sqrt{\epsilon})}$ *running time.*

We know there exists an optimal solution $\mathbf{x}^*$ where the load of each machine belongs to $[1/2, 2]$. Let L_i^* be the load of machine i in $\mathbf{x}^*$ where $1/2 \leq L_i^* \leq 2$. Without loss of generality we further assume that $L_1^* \leq L_2^* \leq \cdots \leq L_m^*$. For some integer $h \geq 1$, let $\mathcal{G}_h$ be the set of jobs whose processing time lies in $[\epsilon(1+\sqrt{\epsilon})^{h-1}, \epsilon(1+\sqrt{\epsilon})^h)$. Given that $\epsilon \leq p_j \leq 1$, every job belongs to some set $\mathcal{G}_h$ for $h \in \{1, \ldots, \tau\}$, where $\tau = \tilde{O}(1/\sqrt{\epsilon})$. For simplicity, we call a job in $\mathcal{G}_h$ a $\mathcal{G}_h$-job. The following structural result contains the key observation for the existence of a PTAS with subexponential time.

LEMMA 2.3. *There exists a feasible solution* $\hat{x}$ *satisfying: i) its objective value is at most* $(1+O(\epsilon))OPT$*, and ii) the machines can be ordered from* 1 *to* m *such that for any* $1 \leq i \leq m-1$ *and* h*, the processing time of every* $\mathcal{G}_h$*-job on machine* $i+1$ *is at most the processing time of any* $\mathcal{G}_h$*-job on machine* i.

Proof. Given an optimal solution x^*, we construct $\hat{x}$ as follows. For machine m, we replace all $\mathcal{G}_h$-jobs with the same number of the smallest $\mathcal{G}_h$-jobs. For machine $m-1$, we replace all $\mathcal{G}_h$-jobs with the same number of the

remaining smallest $\mathcal{G}_h$-jobs, etc. Eventually, every $\mathcal{G}_h$-job on machine $i+1$ is no greater than any $\mathcal{G}_h$-job on machine i. Let $\hat{L}_i = L_i^* + \Delta_i$ be the new load of machine i.

By the definition of $\mathcal{G}_h$, we know that the largest $\mathcal{G}_h$-job has a processing time at most $1+\sqrt{\epsilon}$ times the smallest one. This implies that $L_i^*/(1+\sqrt{\epsilon}) \leq \hat{L}_i \leq (1+\sqrt{\epsilon})L_i^*$, and hence $|\Delta_i| \leq \sqrt{\epsilon}L_i^* \leq 2\sqrt{\epsilon}$. In order to bound the objective function, first write $\sum_{i=1}^m (L_i^* + \Delta_i)^q = \sum_{i=1}^m L_i^{*q}(1+\Delta_i/L_i^*)^q$. Using a Taylor expansion of order 1 on the function $(1+x)^q$ around $x=0$, we obtain that for some $0 \leq \xi_i \leq \Delta_i/L_i^* \leq 1$,

$$\begin{aligned}
\sum_{i=1}^m (L_i^* + \Delta_i)^q &= \sum_{i=1}^m L_i^{*q}\left(1 + q\frac{\Delta_i}{L_i^*} + \frac{q(q-1)}{2}(1+\xi_i)^{q-2}\left(\frac{\Delta_i}{L_i^*}\right)^2\right) \\
&\leq (1+O(\epsilon))\sum_{i=1}^m L_i^{*q} + q\sum_{i=1}^m \Delta_i L_i^{*q-1} \\
&= (1+O(\epsilon))OPT + qL_1^{*q-1}\sum_{k=1}^m \Delta_k + q\sum_{i=2}^m \left[(L_i^{*q-1} - {L_{i-1}^*}^{q-1})\sum_{k=i}^m \Delta_k\right] \\
&\leq (1+O(\epsilon))OPT.
\end{aligned}$$

The last equality uses Abel's transformation (summation by parts). The last inequality follows since, for each i, it holds that $\sum_{k=i}^m \Delta_k \leq 0$, as the last $m-i$ machines received the smallest $\mathcal{G}_h$-jobs for each h. ∎

Exploiting the ordering of the jobs and machines given by Lemma 2.3, we are able to develop a dynamic programming based algorithm to prove Lemma 2.2, see Appendix A.2.

3 Lower Bound

In this section, we will prove the following theorem.

THEOREM 3.1. *Let $q > 1$ be an arbitrary constant. Assuming ETH, there is no PTAS for $P||\sum_i C_i^q$ that runs in $2^{O((1/\varepsilon)^{1/2-\delta})} + n^{O(1)}$ time for any constant $\delta > 0$.*

For the proof we give a fine-grained reduction from a variant of Max3SAT, called 3SAT′ (which we elaborate in the following subsection), to $P||\sum_i C_i^q$.

3.1 3SAT′ - Max3SAT with a Special Structure We study a variant of 3SAT, which we call 3SAT′, whose instances have the following structure: There are n variables $z_1, \ldots, z_n$, where n is a multiple of 3. There are $4n/3$ clauses, such that the set of clauses can be divided into two disjoint sets C_1 and C_2 such that:

- In C_1, every clause is a disjunction (OR operator) of three literals. For each variable z_i, exactly one literal in C_1 belongs to $\{z_i, \neg z_i\}$.
- In C_2, every clause is of the form $z_i \oplus \neg z_k$, where $\oplus$ denotes the XOR operator. Also, for every variable z_i, each literal z_i and $\neg z_i$ appears exactly once within C_2.

For example, $C_1 = \{(z_1 \vee \neg z_2 \vee z_3)\}$ and $C_2 = \{(z_1 \oplus \neg z_2), (z_2 \oplus \neg z_3), (z_3 \oplus \neg z_1)\}$ defines a 3SAT′ instance for $n=3$. Let $cl_2, cl_3, \cdots, cl_{n-1}$ be the clauses in C_1. By re-indexing we can assume that cl_ℓ is of the form $(w_{\ell-1} \vee w_\ell \vee w_{\ell+1})$, where $w_j \in \{z_j, \neg z_j\}$ for all j. Also notice that $|C_1| = n/3$ and $|C_2| = n$. Since every literal appears exactly once in C_2, we define a permutation $\tau : \mathbb{Z}_n \to \mathbb{Z}_n$ (i.e. a bijection) such that $\tau(i) = k$ for each $(z_i \oplus \neg z_k) \in C_2$.

Similarly to 3SAT, it is also difficult to distinguish instances of 3SAT′ where almost all clauses are satisfiable and instances where at most certain fraction of the clauses can be satisfied, as implied by the following lemma. See Appendix B for its proof.

LEMMA 3.1. *Assuming ETH, there exists a constant $\beta \in (0,1)$ such that for any sufficiently small $\epsilon', \delta > 0$ there is no algorithm with running time $2^{O(n^{1-\delta})}$ that distinguishes between instances of 3SAT′ with $4n/3$ clauses where at least $(1-\epsilon')\cdot 4n/3$ clauses are satisfiable, from instances where at most $(\beta+\epsilon')\cdot 4n/3$ clauses are satisfiable.*

3.2 Overview of the reduction We now briefly describe the structure of the constructed scheduling instance. The detailed reduction will be presented in Appendix F. We remark that the high-level structure of the scheduling instance resembles the classical reduction and that of [7]. New technical ingredients are in job processing times, as we will elaborate in Section 3.3.

For an instance I_{sat} of 3SAT$'$ with n variables, we construct the following 6 kinds of jobs:

- Variable jobs: For each positive (or negative, resp.) literal, say, z_i (or $\neg z_i$, resp.), two pairs of variable jobs $V^{\rho}_{i,+,1}$ and $V^{\rho}_{i,+,2}$ (or $V^{\rho}_{i,-,1}$ and $V^{\rho}_{i,-,2}$, resp.) are constructed where $\rho \in \{T, F\}$. In total, we construct 4 jobs for each (positive or negative) literal, i.e., 8 jobs for each variable.
- Clause jobs: For each clause cl_ℓ of C_1, one clause job CL^T_ℓ and two copies of clause job CL^F_ℓ are constructed. Recall that $|C_1| = n/3$, we construct n clause jobs.
- Truth-assignment jobs, link jobs and dummy jobs: These three kinds of jobs will be created suitably so that the conditions below (CO1 to CO4) are satisfied.
- Gap jobs: Let Q be a target makespan. We construct $\tilde{O}(n)$ gap jobs and the same number of machines to create gaps. Roughly speaking, every feasible schedule whose objective value is not too large will have one gap job on each machine, leaving a gap that must be filled up such that the load of the machine is exactly Q. We will create 4 kinds of gaps (incurred by gap jobs) satisfying the following conditions:
  - CO1. Variable-Truth gaps. To fill up these gaps, for any i either $V^F_{i,+,1}$, $V^F_{i,+,2}$, $V^T_{i,-,1}$, $V^T_{i,-,2}$, or $V^T_{i,+,1}$, $V^T_{i,+,2}$, $V^F_{i,-,1}$, $V^F_{i,-,2}$ are used. Truth-assignment jobs are created for this purpose.
  - CO2. Variable-Clause-Dummy gaps. For each clause $cl_\ell \in C_1$, there are three variable-clause-dummy gaps. If the positive (or negative, resp.) literal z_i (or $\neg z_i$, resp.) is in $cl_\ell \in C_1$, then a variable-clause-dummy gap is created so that it could only be filled up by CL^ρ_ℓ and $V^{\rho'}_{i,+,1}$ (or CL^ρ_ℓ and $V^{\rho'}_{i,-,1}$, resp.), where $\rho, \rho' \in \{T, F\}$, together with a dummy job. Further, the gap ensures that CL^T_ℓ has to be scheduled with either $V^T_{i,+,1}$ or $V^T_{i,-,1}$.
  - CO3. Variable-Link and Link-Link gaps. For each clause $(z_i \oplus \neg z_k) \in C_2$ we create a collection of Variable-Link and Link-Link gaps. To fill up these gaps, either $V^T_{i,+,2}$ and $V^F_{k,-,2}$, or $V^F_{i,+,2}$ and $V^T_{k,-,2}$ are used. Link jobs are created for this purpose (see Section 3.3 for more details on this construction).
  - CO4. Variable-Dummy gaps. Recall that 8 variable jobs are constructed for a variable and only 7 of them are used for the 4 kinds of gaps above (either $V^\rho_{i,+,1}$ or $V^\rho_{i,-,1}$ is left, where $\rho \in \{T, F\}$), the remaining one together with a dummy job will be used to fill these gaps.

With this construction, it is not difficult to verify that if every gap is filled exactly, I_{sat} is satisfiable. To see why, if $V^F_{i,+,1}$, $V^F_{i,+,2}$, $V^T_{i,-,1}$, $V^T_{i,-,2}$ are used in the variable-truth gaps, then we let variable z_i be true, otherwise we let it be false. For any clause of C_1, say, cl_ℓ, there is one CL^T_ℓ and it must be scheduled with a true variable job, say, $V^T_{i,+,1}$ if z_i is a literal in cl_ℓ (or $V^T_{i,-,1}$ if $\neg z_i$ is a literal in cl_ℓ). If $V^T_{i,+,1}$ (or $V^T_{i,-,1}$, resp.) is scheduled with CL^T_ℓ, then the positive (or negative, resp.) literal z_i (or $\neg z_i$, resp.) is in cl_ℓ. Meanwhile the variable z_i is true (or false, resp.) since otherwise $V^T_{i,+,1}$ (or $v^T_{i,-,1}$,resp.) are used to fill variable-truth gaps. Thus clause cl_ℓ is satisfied. For any clause of C_2, say, $(z_i \oplus \neg z_k)$, if $V^T_{i,+,2}$ and $V^F_{k,-,2}$ (or $V^F_{i,+,2}$ and $V^T_{k,-,2}$, resp.) are used to fill up the corresponding variable-link and link-link gaps, then variables z_i and z_k are both true (false, resp.) since otherwise $V^T_{i,+,2}$ and $V^F_{k,-,2}$ ($V^F_{i,+,2}$ and $V^T_{k,-,2}$, resp.) would have been used to fill up the variable-truth gaps. Hence, $(z_i \oplus \neg z_k)$ is satisfied. Similarly, if I_{sat} is satisfiable, then every gap can be filled up.

Chen et al. [7] provided a reduction that meets the above requirement with job processing times, and hence the target value Q, being $O(n^{1+\delta})$ for arbitrarily small constant $\delta > 0$. Unfortunately, using this reduction we can only deduce a weaker lower bound of $2^{O((1/\epsilon)^{1/3-\delta})}$ (see Appendix C for a detailed discussion). For our purpose, we need to design job processing times to achieve a stronger *ratio-preserving* property, as we elaborate below.

Recall that we are given an instance of 3SAT$'$ with n variables and $4n/3$ clauses. For a given solution, we say a machine is *good* if its load is exactly Q (in which case there is exactly one gap job on it and the gap is filled up exactly), and is *bad* otherwise (in which case its load is at least $Q + 1/2$ or at most $Q - 1/2$). Our scheduling instance will additionally satisfy the following properties:

i. There are $m = \tilde{O}(n)$ machines and the target makespan is $Q = \tilde{O}(n)$.

ii. Each processing time is a multiple of $1/2$ and the total job processing time equals mQ.

iii. Conditions CO1 to CO4 are satisfied. Additionally, the following ratio-preserving properties are satisfied. For any $\vartheta \in (0,1)$ it holds that:

- If the 3SAT$'$ instance admits a truth assignment where at most ϑn clauses are not satisfied, then the constructed scheduling instance admits a feasible solution with at most $\vartheta_1 n$ bad machines, for some $\vartheta_1 = \Theta(\vartheta)$. In particular, the load of these bad machines is $Q+1$ or $Q-1$.
- If any truth assignment for the 3SAT$'$ instance has at least ϑn clauses that are not satisfied, then in any feasible schedule of the constructed scheduling instance there are at least $\vartheta_2 n$ bad machines, for some $\vartheta_2 = \Theta(\vartheta)$.

Before giving more details of the construction, we briefly argue that an instance satisfying properties (i)-(iii) implies Theorem 3.1.

Proof. [Proof Idea (Theorem 3.1)] Take $q = 2$ for simplicity. We assume by contradiction that there exist some sufficiently small $\delta > 0$, such that for any $\epsilon > 0$ there is an $(1+\epsilon)$-approximation algorithm with running time $2^{O((1/\epsilon)^{1/2-\delta})} + n^{O(1)}$. Let $\beta \in (0,1)$ be a constant, and let $\epsilon', \delta > 0$ be sufficiently small numbers, as in Lemma 3.1. We show that, for an appropriately chosen ϵ, the PTAS can be used to distinguish, in time $2^{n^{1-\delta}}$, 3SAT$'$ instances where at least $(1-\epsilon') \cdot 4n/3$ clauses are satisfiable, from 3SAT$'$ instances where at most $(\beta+\epsilon') \cdot 4n/3$ clauses are satisfiable, contradicting ETH by Lemma 3.1. Indeed, we first observe that every bad machine will cause the objective value to increase by at least some fixed constant. A straightforward but crucial observation follows from the fact that, for load balancing problems, the total difference from the average load is 0. That is, if $C_i = Q + \Delta_i$, then the cost is

$$\sum_{i=1}^{m} C_i^2 = \sum_{i=1}^{m} (Q+\Delta_i)^2 = \sum_{i=1}^{m} (Q^2 + 2Q\Delta_i + \Delta_i^2) = mQ^2 + \sum_{i=1}^{m} \Delta_i^2, \tag{3.1}$$

where the last equality follows as $\sum_i \Delta_i = 0$ (for general $q > 1$, a similar statement follows from a Taylor expansion, as in the proof of Lemma 2.3). Consequently, if at least $(1-\epsilon') \cdot 4n/3$ clauses of I_{sat} are satisfiable, then at most $\Theta(\epsilon' n)$ machines will have a load of either $Q+1$ or $Q-1$, and hence the optimal objective value of the constructed scheduling instance is at most $mQ^2 + \Theta(\epsilon' n)$ by Eq (3.1). On the other hand, if at most $(\beta+\epsilon') \cdot 4n/3$ clauses of I_{sat} are satisfiable for some constant $\beta < 1$, then $\Delta_i \geq 1/2$ for at least $\Theta((1-\beta-\epsilon')n)$ machines. By Eq (3.1) the optimal objective value of the constructed scheduling instance is at least $mQ^2 + \Theta((1-\beta-\epsilon')n) = mQ^2 + \Theta(n)$ (see Lemma G.20 for the detailed computation). Now we apply the efficient PTAS with $\epsilon = \Theta(\frac{n^{1-\delta}}{mQ^2}) \approx \Theta(1/n^{2+\delta})$. Given the fact that $mQ^2\epsilon = \Theta(n^{1-\delta})$, if at least $(1-\epsilon') \cdot 4n/3$ clauses of I_{sat} are satisfiable, then the PTAS should return a schedule with objective value at most $mQ^2 + \Theta(\epsilon' n) + \Theta(n^{1-\delta}) = mQ^2 + \Theta(\epsilon' n)$. Otherwise, the PTAS returns a schedule with objective value at least $mQ^2 + \Theta(n)$. Theorem 3.1 follows as our PTAS has a running time of $2^{O((1/\epsilon)^{1/2-\delta})} + n^{O(1)} \leq 2^{O(n^{1-\delta})}$. ∎

Remark. One can verify that if Q is larger, e.g., $Q = \Theta(n^2)$, then the above argument only rules out a PTAS of running time $2^{O((1/\epsilon)^{1/4-\delta})}$. Hence, simultaneously enforcing the ratio-preserving property while having $Q = \tilde{O}(n)$ is the main technical challenge, which we overcome with our new number-theoretic constructions, as we elaborate in the following.

The rest of the paper is organized as follows. In Section 3.3 we give an overview of the main technical ingredients for the construction of the processing times in our reduction. We also motivate our number theoretical constructions, which are specified in Section 3.4. In Appendix F we present the complete reduction. In Appendix G we show its correctness and conclude Theorem 3.1.

3.3 Defining Processing times: Main Techniques To illustrate the main technical ingredient, in the following part of this subsection we will focus on conditions CO2 and CO3 while ignoring the other conditions (which can be handled using the techniques for CO2 and CO3). Recall that our goal is to create suitable gap jobs that can only be filled up by specific jobs.

We can view each job, say, $V^T_{i,+,1}$, as a combination of three components – the type-component $V_{\cdot,+,1}$ (indexed by i), the index-component i, and the T/F-component T. Ignoring dummy jobs for simplicity, conditions CO2 and CO3 involve 5 different type-components, including $V_{\cdot,+,1}$, $V_{\cdot,+,2}$ $V_{\cdot,-,1}$ $V_{\cdot,-,2}$ and CL.. Denote by $s(\cdot)$ the processing time of a job. We can define the processing time of a job into a summation of three terms corresponding to components, e.g., $s(V^T_{i,+,1}) = \mu(V_{\cdot,+,1})+\sigma(i)+\eta(T)$, where the functions μ, σ, η map the type-component, index-component and T/F-component of a job to some positive integers. Now the question becomes: how can we define functions μ, σ, η such that from their sum, e.g., $\mu(V_{\cdot,+,1}) + \mu(\text{CL.}) + \sigma(i) + \sigma(\ell) + \eta(T) + \eta(F)$, we can conclude that it can only be added up by $s(V^{\rho}_{i',+,1})$ and $s(\text{CL}^{\rho'}_{\ell'})$, where $\{i', \ell'\} = \{i, \ell\}$ and $\{\rho, \rho'\} = \{T, F\}$. Notice that there are only a constant number of different type-components and T/F-components, it is thus easy to define μ and η. For example, let σ_{max} be a sufficiently large value that exceeds the maximal value of σ and η, and define $\mu(V_{\cdot,+,1}), \mu(V_{\cdot,+,2}), \mu(V_{\cdot,-,1}), \mu(V_{\cdot,-,2}), \mu(\text{CL.})$ to be $10^5\sigma_{max}, 10^4\sigma_{max}, 10^3\sigma_{max}, 10^2\sigma_{max}, 10\sigma_{max}$, then from the sum $\mu(V_{\cdot,+,1}) + \mu(\text{CL.})$ it is very easy to identify the type-components of two jobs.

The main difficulty lies in the function σ as we require job processing times to be $\tilde{O}(n)$, whereas σ must map $[n]$ to $[\tilde{O}(n)]$ such that

$$\sigma(i) + \sigma(\ell) = \sigma(i') + \sigma(\ell') \implies \{i, \ell\} = \{i', \ell'\}. \tag{3.2}$$

In other words, we require the sum $\sigma(i) + \sigma(\ell)$ to be unique among the sums of all possible pairs (i, ℓ). Recall the special structure of 3SAT′, where each clause $cl_\ell \in C_1$ is of the form $(w_{\ell-1} \vee w_\ell \vee w_{\ell+1})$ such that $w_j \in \{z_j, \neg z_j\}$ for all j. Hence, for condition CO2, it suffices to guarantee Eq (3.2) for $i \in \{\ell - 1, \ell, \ell + 1\}$. Recall that a Salem-Spencer set is a set of numbers where no three of which form an arithmetic progression, hence if we let σ map $[n]$ to a Salem-Spencer set of size n, Eq (3.2) always holds for $\ell = i$. For our purpose, we need to generalize the construction of a Salem-Spencer set such that in addition to $2\sigma(\ell)$, the sum of any two adjacent numbers $\sigma(\ell) + \sigma(\ell + 1)$ is also unique, as we show in Lemma 3.3.

Condition CO3 is more complicated, as each clause of C_2 is of the form $(z_i \oplus \neg z_k)$ with $k = \tau(i)$, where the permutation τ is arbitrary. If we consider the index-components of the two variable jobs $V^{\rho}_{i,+,2}$ and $V^{\rho'}_{k,-,2}$, we cannot guarantee that $\sigma(i) + \sigma(k) = \sigma(i') + \sigma(k')$ implies $\{i, k\} = \{i', k'\}$. Consider the following indirect approach. Suppose for each $(z_i \oplus \neg z_k)$ we can construct a pair of jobs $\text{LN}^T_{(i,k)}$ and $\text{LN}^F_{(i,k)}$ (called link jobs), and meanwhile create two gaps such that they must be filled up by $V^{\rho_1}_{i,+,2}$ together with $\text{LN}^{\rho'_1}_{(i,k)}$, and $V^{\rho_2}_{k,-,2}$ together with $\text{LN}^{\rho'_2}_{(i,k)}$ respectively, and furthermore, $\{\rho_1, \rho'_1\} = \{\rho_2, \rho'_2\} = \{T, F\}$, then we know that if both gaps are filled up, then either $V^T_{i,+,2}$ and $V^F_{k,-,2}$, or $V^F_{i,+,2}$ and $V^T_{k,-,2}$ are used, which is sufficient for condition CO3. Using this idea, instead of designing σ such that the sum $\sigma(i) + \sigma(k)$ is unique, we seek to design σ such the pair (i, k) is "uniquely -linked" in the sense that there exists some number $e_i = \tilde{O}(n)$ such that the sums $\sigma(i)+e_i$ and $\sigma(k)+e_i$ are both unique among the sums of all pairs. Unfortunately, requiring the uniqueness of $\sigma(i) + e_i$ and $\sigma(k) + e_i$ is still too strong. We will show in Lemma 3.4 that for every $k = \tau(i)$ there exists a sequence of $\omega = O(\frac{\log n}{\log \log n})$ numbers $e_{i,1}$, $e_{i,2}$, …, $e_{i,\omega}$ such that the sums $\sigma(i) + e_{i,1}$, $e_{i,1} + e_{i,2}$, …, $e_{i,\omega-1} + e_{i,\omega}$, $e_{i,\omega} + \sigma(k)$ are all unique. Consequently, instead of creating one pair of link jobs, we will create ω pairs of link jobs for each $(z_i \oplus \neg z_k)$, ensuring condition CO3.

3.4 Set of Integers with Unique Adjacent Sum and Linked Sum In this section, we present our main technical contribution regarding the number-theoretic constructions needed in our reduction.

Some notation. Recall that we let $\mathbb{Z}_n = \{1, 2, \cdots, n\}$. All the logarithms are taken with base e unless stated otherwise. We will use $\cdot$ in the subscript to denote an arbitrary index, e.g., $x_\cdot$ refers to x_i for some i. We write vectors in boldface, e.g. $\mathbf{x}, \mathbf{y}$. Vectors start with its 0-th coordinate. For any υ-dimensional vector $\mathbf{c}$, $\mathbf{c}[h]$ denotes its h-th coordinate for $0 \le h \le \upsilon - 1$, and $\mathbf{cx} = \sum_{h=0}^{\upsilon-1} \mathbf{c}[h]x^h$.

LEMMA 3.2. *Let $N \in \mathbb{Z}^+$. There exists a subset $\mathcal{S} \subseteq \mathbb{Z}_N$ such that $|\mathcal{S}| \ge N^{1-c_0\sqrt{\frac{1}{\log N}}}$ for some sufficiently large c_0 (in particular, $c_0 \ge 7$ suffices), and for any $y \in \mathcal{S}$ and $1 \le h \le 5$, the linear equation $h \cdot y = y_1 + y_2 + \cdots + y_h$ with $y_i \in \mathcal{S}$ for all i has a unique solution $y_1 = y_2 = \cdots = y_h = y$.*

The proof of Lemma 3.2 mainly utilizes the idea for constructing Salem–Spencer sets [3] and can be found in Appendix D. In particular, we can show that $|\mathcal{S}| \ge N^{1-7\sqrt{\frac{1}{\log N}}}$. For any integer $d \in \mathbb{Z}^+$, we denote by $\mathcal{S}_d$

the subset of $\mathbb{Z}_d$ that satisfies Lemma 3.2. Now we are ready to prove Lemma 3.3, which is one of our two main number-theoretical results.

LEMMA 3.3. *Let $N \in \mathbb{Z}^+$, $d = \lceil e^{(\sqrt{\log\log N}+c)^2} \rceil = e^{\mathcal{O}(\sqrt{\log\log N})} \cdot \log N$ for some sufficiently large c ($c \geq 7$ suffices) and $x = 5d+1$. There exists an injection $\sigma : \mathbb{Z}_N \to \mathbb{Z}_{N'}$ satisfying the following 4 properties:*

1. *$N' = N^{1+O(\frac{1}{\sqrt{\log\log N}})}$;*
2. *For any $i \in \mathbb{Z}_N$, $\sigma(i) = \sum_{j=0}^{\gamma} \mathbf{a}_i[j] x^j$ for some $\mathbf{a}_i[j] \in \mathcal{S}_d$, $0 \leq j \leq \gamma$, where $\gamma = \lceil \frac{\log N}{\log\log N} \rceil + O(\frac{\log N}{(\log\log N)^{3/2}})$;*
3. *For any $1 \leq h \leq 5$ and $i \in \mathbb{Z}_N$, the equation $h \cdot \sigma(i) = \sigma(i_1) + \sigma(i_2) + \cdots + \sigma(i_h)$, $i_j \in \mathbb{Z}_N$ has a unique solution $i_1 = i_2 = \cdots = i_h = i$. Further, the equation $h \cdot \sigma(i) = \sigma(i_1) + \sigma(i_2) + \cdots + \sigma(i_k)$, $i_j \in \mathbb{Z}_N$ has no feasible solution when $1 \leq k < h$ or $h < k \leq 5$;*
4. *For any $i \leq N-1$, the linear equation $\sigma(i) + \sigma(i+1) = \sigma(i_1) + \sigma(i_2)$, $i_1 \leq i_2$ has a unique solution $i_1 = i$, $i_2 = i+1$. Furthermore, the linear equation $\sigma(i) + \sigma(i+1) = \sigma(i_1) + \sigma(i_2) + \cdots + \sigma(i_k)$ has no feasible solution when $k = 1$ or $2 < k \leq 5$.*

Proof. By Lemma 3.2 we know $|\mathcal{S}_d| \geq d^{1-c_0\sqrt{\frac{1}{\log d}}}$ for some constant c_0 (in particular, we can choose $c_0 = 7$). Let $\hat{\mathcal{S}} \subseteq \mathcal{S}_d$ be an arbitrary subset such that $|\hat{\mathcal{S}}| = 2^\omega$ for some integer ω such that $|\hat{\mathcal{S}}| \geq 1/2 \cdot |\mathcal{S}_d|$, then it is easy to see that $|\hat{\mathcal{S}}| = d^{1-\Theta(\sqrt{\frac{1}{\log d}})}$. Consider all the integers that can be written as $\sum_{i=0}^{\beta} \mathbf{a}[i]x^i = \mathbf{a} \cdot \mathbf{x}$ for some integer β, $\mathbf{a} = (\mathbf{a}[0], \mathbf{a}[1], \cdots, \mathbf{a}[\beta]) \in \hat{\mathcal{S}}^{\beta+1}$, and $\mathbf{x} = (1, x, \cdots, x^\beta)$, for $x = 5d+1$. It is easy to see that we obtain $|\hat{\mathcal{S}}|^{\beta+1}$ different integers constructed this way.

Simple calculations show that $|\hat{\mathcal{S}}|^{\beta+1} \geq N$ if

$$\beta \geq \frac{\log N}{\log d}\left(1 + \Theta\left(\sqrt{\frac{1}{\log d}}\right)\right).$$

Hence, by picking $\beta = \lceil \frac{\log N}{\log\log N} \rceil + O(\frac{\log N}{(\log\log N)^{3/2}})$, we can guarantee that $|\hat{\mathcal{S}}|^{\beta+1} \geq N$. For $d \geq e^{(\sqrt{\log\log N}+7)^2}$, we notice that $\sqrt{\log d} \geq \sqrt{\log\log N} + 7$, hence

$$|\mathcal{S}_d| \geq d^{1-7\sqrt{\frac{1}{\log d}}} = e^{\log d - 7\sqrt{\log d}} > e^{\log\log N} = \log N > \beta + 1.$$

Hence, we can define an arbitrary injection g that maps $j \in \{0, 1, \cdots, \beta\}$ to a distinct number in $\mathcal{S}_d$.

Consider all the vector $\mathbf{a}$'s. For any two vectors $\mathbf{a}_j$ and $\mathbf{a}_k$, we say they are *close* if $\mathbf{a}_j$ and $\mathbf{a}_k$ differ by exactly one coordinate, i.e., there exists some $0 \leq j^* \leq \beta$ such that $\mathbf{a}_i[j] = \mathbf{a}_k[j]$ for all $j \neq j^*$ and $\mathbf{a}_i[j^*] \neq \mathbf{a}_k[j^*]$. We claim the following.

CLAIM 1. *Vectors in $\hat{\mathcal{S}}^{\beta+1}$ can be ordered such that any two consecutive vectors are close.*

Proof. Recall that $|\hat{\mathcal{S}}| = 2^\omega$, hence we can map each $a_j \in \hat{\mathcal{S}}$ to a distinct ω-bit binary number (or more specifically, a binary string) within $\{0,1\}^\omega$. Let $\xi : \hat{\mathcal{S}} \to \{0,1\}^\omega$ be an arbitrary one-to-one mapping, then we can define an extended mapping $\xi' : \hat{\mathcal{S}}^{\beta+1} \to \{0,1\}^{(\beta+1)\omega}$ such that $\mathbf{a}$ is mapped to a $(\beta+1)\omega$-bit binary string $\xi(\mathbf{a}[0])\xi(\mathbf{a}[1])\cdots\xi(\mathbf{a}[\beta])_2$. If we can order all $(\beta+1)\omega$-bit binary string such that every adjacent numbers differ by exactly one bit, then the inverse of these binary strings gives a sequence of $\mathbf{a}$'s such that adjacent vectors are close.

Now we prove the following statement: for any $n \in \mathbb{Z}$, $n \geq 2$ and an arbitrary string $b \in \{0,1\}^n$, all binary strings of $\{0,1\}^n$ can be ordered in a sequence starting with b such that any two adjacent strings differ by exactly one bit. We show this by induction. The statement is clearly true for $n = 2$. Suppose it is true for all $n \leq n'$, we prove it also holds for $n = n' + 1$. Consider the first bit of b, which can be 0 or 1. Assume it is 1 (the case of 0 can be proved in a similar way), then $b = 1b_1$ for some $b_1 \in \{0,1\}^{n'}$. According to the induction hypothesis, all binary strings of $\{0,1\}^{n'}$ can be ordered in a sequence starting with b_1 such that any two adjacent binary strings only differ by one bit. Let such a sequence be $b_1, b_2, \cdots, b_{2^{n'}}$, then all binary strings of $\{0,1\}^{n'+1}$ can be ordered as $1b_1, 1b_2, \cdots, 1b_{2^{n'}}, 0b_{2^{n'}}, 0b_{2^{n'}-1}, \cdots, 0b_1$. Hence, the statement is true, and Claim 1 follows. ∎

Now consider an arbitrary ordering of vectors of $\hat{\mathcal{S}}^{\beta+1}$ that satisfies Claim 1. Let the sequence be $\mathbf{a}_1, \mathbf{a}_2, ...$ where $\mathbf{a}_i = (\mathbf{a}_i[0], \mathbf{a}_i[1], \cdots, \mathbf{a}_i[\beta])$ denote the i-th vector in the sequence. Recall that each $\mathbf{a}_i$ is a $(\beta+1)$-dimensional vector. Let $prec(i)$ be the unique coordinate where $\mathbf{a}_i$ and $\mathbf{a}_{i-1}$ differ. Similarly, let $succ(i)$ be the coordinate where $\mathbf{a}_i$ and $\mathbf{a}_{i+1}$ differ. By definition it holds that $prec(i+1) = succ(i)$. For the first and last vectors in the sequence, we define additionally that $\mathbf{a}_1[prec(1)] = \mathbf{a}_{|\hat{\mathcal{S}}^{\beta+1}|}[succ(|\hat{\mathcal{S}}^{\beta+1}|)] = 0$. Let $\gamma = \beta + 8$ and recall the injection $g : \{0, 1, \cdots, \beta\} \to \mathcal{S}_d$. We define σ such that

$$\sigma(i) = \mathbf{a}_i \cdot \mathbf{x} + \mathbf{a}_i[prec(i)]x^{\beta+1} + \mathbf{a}_i[succ(i)]x^{\beta+2} + g\left(prec(i)\right)x^{\beta+5} + g\left(succ(i)\right)x^{\beta+6}, \quad \text{if } i \text{ is odd;}$$
$$\sigma(i) = \mathbf{a}_i \cdot \mathbf{x} + \mathbf{a}_i[prec(i)]x^{\beta+3} + \mathbf{a}_i[succ(i)]x^{\beta+4} + g\left(prec(i)\right)x^{\beta+7} + g\left(succ(i)\right)x^{\beta+8}, \quad \text{if } i \text{ is even.}$$

where $\mathbf{x} = (1, x, x^2, \cdots, x^\beta)$, while noting that $prec(i), succ(i) \leq \beta < x$. Also, remark that all coeficients in the polynomial expression belong to $\mathcal{S}_d$. Given that $x = 5d+1$ and $\beta = \frac{\log N}{\log d}\left(1 + \Theta\left(\sqrt{\frac{1}{\log d}}\right)\right)$, one can verify that

$$\sigma(i) \leq (5d+1)^{\beta+9} = e^{(\beta+9)\log(5d+1)} \leq e^{\left(\frac{\log N}{\log d} + \mathcal{O}\left(\frac{\log N}{(\log d)^{3/2}}\right)\right)(\log d + \mathcal{O}(1))} \leq N' = N^{1+O(\frac{1}{\sqrt{\log\log N}})},$$

for any $i \leq N$, hence properties 1,2 of Lemma 3.3 hold.

Consider the equation $h \cdot \sigma(i) = \sigma(i_1) + \sigma(i_2) + \cdots + \sigma(i_k)$ for $k, h \leq 5$. The right-hand side of this equation can be expressed as $\sigma(i_1) + \sigma(i_2) + \cdots + \sigma(i_k) = \sum_{j=1}^{\gamma} b_j x^j$, for some coefficients b_j. Notice as each coefficient $\mathbf{a}_i[j]$ belongs to $\hat{\mathcal{S}} \subseteq \mathcal{S}_d$, which is at most $\mathbf{a}_i[j] \leq d < x/5$, then each b_h is a sum of at most $k \leq 5$ such coefficients, and thus $0 \leq b_h < x$. We obtain a similar statement for the left-hand side. Hence the coefficients of terms of the same degree must coincide, and we have $h \cdot \mathbf{a}_i[j] = \sum_{h=1}^{k} \mathbf{a}_{i_h}[j]$ for all $0 \leq j \leq \beta$. According to Lemma 3.2, we know the only solution for the above is $k = h$ and $i_1 = i_2 = \cdots = i_k = i$, hence property 3 is true.

It remains to prove property 4. We suppose i is odd in the following; the case of i being even can be proved analogously. Consider $\sigma(i) + \sigma(i+1)$ which is equal to

$$\sum_{j=0}^{\beta} b_j x^j + \mathbf{a}_i[prec(i)]x^{\beta+1} + \mathbf{a}_i[succ(i)]x^{\beta+2} + \mathbf{a}_{i+1}[prec(i+1)]x^{\beta+3} + \mathbf{a}_{i+1}[succ(i+1)]x^{\beta+4}$$
$$+ \quad g\left(prec(i)\right)x^{\beta+5} + g\left(succ(i)\right)x^{\beta+6} + g\left(prec(i+1)\right)x^{\beta+7} + g\left(succ(i+1)\right)x^{\beta+8}.$$

As before, the coefficients of terms of the same degree must coincide. We know that $\mathbf{a}_i$ and $\mathbf{a}_{i+1}$ only differs at coordinate $succ(i) = prec(i+1)$, hence $b_j = 2\mathbf{a}_i[j]$ for $j \neq succ(i)$. If $\sigma(i_1) + \sigma(i_2) = \sigma(i) + \sigma(i+1)$, we know $\mathbf{a}_{i_1}[j] + \mathbf{a}_{i_2}[j] = 2\mathbf{a}_i[j]$ for $j \neq succ(i)$, and by the fact that $\mathbf{a}_k[j] \in \mathcal{S}_d$ we know it must hold that

$$\mathbf{a}_{i_1}[j] = \mathbf{a}_{i_2}[j] = \mathbf{a}_i[j] \quad \text{for all } j \neq succ(i). \tag{3.3}$$

Now consider the $succ(i)$-th coordinate. We know that among i_1 and i_2 one is even and one is odd, for otherwise in $\sigma(i_1) + \sigma(i_2)$ either the coefficients of $x^{\beta+1}$ and $x^{\beta+2}$ are 0, or the coefficients of $x^{\beta+3}$ and $x^{\beta+4}$ are 0. In either case, this means that $\mathbf{a}_i$ or $\mathbf{a}_{i+1}$ has a 0 coefficient, which is a contradiction as $\mathcal{S}_d \subseteq \mathbb{Z}_d$. Let $\{i_o, i_e\} = \{i_1, i_2\}$ where i_o is odd and i_e is even. Then from $\sigma(i_1) + \sigma(i_2) = \sigma(i) + \sigma(i+1)$, we have

$$\begin{aligned}
& \mathbf{a}_{i_o}[prec(i_o)]x^{\beta+1} + \mathbf{a}_{i_o}[succ(i_o)]x^{\beta+2} + \mathbf{a}_{i_e}[prec(i_e)]x^{\beta+3} + \mathbf{a}_{i_e}[succ(i_e)]x^{\beta+4} + \\
+ \quad & g\left(prec(i_o)\right)x^{\beta+5} + g\left(succ(i_o)\right)x^{\beta+6} + g\left(prec(i_e)\right)x^{\beta+7} + g\left(succ(i_e)\right)x^{\beta+8} \\
= \quad & \mathbf{a}_i[prec(i)]x^{\beta+1} + \mathbf{a}_i[succ(i)]x^{\beta+2} + \mathbf{a}_{i+1}[prec(i+1)]x^{\beta+3} + \mathbf{a}_{i+1}[succ(i+1)]x^{\beta+4} + \\
+ \quad & g\left(prec(i)\right)x^{\beta+5} + g\left(succ(i)\right)x^{\beta+6} + g\left(prec(i+1)\right)x^{\beta+7} + g\left(succ(i+1)\right)x^{\beta+8}.
\end{aligned}$$

Now we can deduce that $\mathbf{a}_{i_o}[succ(i_o)] = \mathbf{a}_i[succ(i)]$ and $succ(i_o) = succ(i)$ (since g is an injection). Using Equation (3.3) we conclude that $\mathbf{a}_{i_o} = \mathbf{a}_i$. Similarly, $\mathbf{a}_{i+1}[prec(i+1)] = \mathbf{a}_{i_e}[prec(i_e)]$, $prec(i+1) = prec(i_e)$. As $succ(i) = prec(i+1)$, Equation (3.3) yields that $\mathbf{a}_{i_e} = \mathbf{a}_{i+1}$.

Each vector in the sequence is unique, so we know $i_o = i$ and $i_e = i+1$. Using that $i_1 \leq i_2$ we obtain that $i_1 = i$ and $i_2 = i+1$. Hence, property 4 is proved. Lemma 3.3 follows. ∎

With Lemma 3.3, we are ready for the main result of this section.

LEMMA 3.4. *Let τ be an arbitrary permutation of $\mathbb{Z}_n$. Then there exists a set $\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$ of positive integers together with an auxiliary set $E = \bigcup_{i=1}^{n} E_i$ of positive integers such that:*

- *All integers in $\mathcal{S} \cup E$ are bounded by $n^{1+O(\frac{1}{\sqrt{\log \log n}})}$;*
- *$E_i = \{e_{i,1}, e_{i,2}, \cdots, e_{i,\omega}\}$ for all i, where $\omega = O(\frac{\log n}{\log \log n})$;*
- *$E_i \cap E_{i'} = \emptyset$ for any $i' \neq i$;*
- *For every $i \in \mathbb{Z}_n$ and $k = \tau(i)$, each sum $s_i + e_{i,1}$, $e_{i,1} + e_{i,2}$, $\cdots$, $e_{i,\omega-1} + e_{i,\omega}$, $e_{i,\omega} + s_k$ is unique, that is, there is no other pair in $\mathcal{S} \cup E$ that adds up to the same value.*

In particular, all these properties are satisfied by setting $s_i = \sigma(i)$ where $\sigma : \mathbb{Z}_N \to \mathbb{Z}_{N'}$ is the function specified in Lemma 3.3 by taking $N = n$.

We start with a natural proof idea. Recall Lemma 3.3, where $\sigma(i) = (\mathbf{a}_i[0], \mathbf{a}_i[1], \cdots, \mathbf{a}_i[\gamma]) \cdot (1, x, \cdots, x^\gamma) = \mathbf{a}_i \cdot \mathbf{x}$ such that $\mathbf{a}_i[j] < x/5$, whereas when we add two values, say, $\sigma(i)+\sigma(k)$, we can directly add each coordinate $\mathbf{a}_i[j]+\mathbf{a}_k[j]$. Given i and k, how can we guarantee that the equation $\mathbf{a}_i+\mathbf{a}_k = (\mathbf{a}_i[0]+\mathbf{a}_k[0], \mathbf{a}_i[1]+\mathbf{a}_k[1], \cdots, \mathbf{a}_i[\gamma]+\mathbf{a}_k[\gamma]) = \mathbf{a}_\ell+\mathbf{a}_r$ has a unique solution $\{\ell, r\} = \{i, k\}$? A simple observation is that, since $\mathbf{a}_i[j] \in \mathcal{S}_d$ (by property 3 of Lemma 3.3), we know that $2\mathbf{a}_i[j]$ can only be expressed as $\mathbf{a}_i[j]+\mathbf{a}_i[j]$, and $\mathbf{a}_i[j]$ can only be expressed as $\mathbf{a}_i[j]+0$. Consequently, we may lift up the dimension by writing $\sigma(i) = (\mathbf{a}_i[0], \mathbf{a}_i[1], \cdots, \mathbf{a}_i[\gamma], 0) \cdot (1, x, \cdots, x^\gamma, x^{\gamma+1})$, and consider the following sequence of numbers:

$$
\begin{array}{lccccccc}
 & (\mathbf{a}_i[0], & \mathbf{a}_i[1], & \cdots, & \mathbf{a}_i[\gamma-1], & \mathbf{a}_i[\gamma], & 0) \\
\to & (0, & \mathbf{a}_i[1], & \cdots, & \mathbf{a}_i[\gamma-1], & \mathbf{a}_i[\gamma], & \mathbf{a}_k[0]) \\
\to & (\mathbf{a}_k[0], & \mathbf{a}_i[1], & \cdots, & \mathbf{a}_i[\gamma-1], & \mathbf{a}_i[\gamma], & 0) \\
\to & (\mathbf{a}_k[0], & 0, & \cdots, & \mathbf{a}_i[\gamma-1], & \mathbf{a}_i[\gamma], & \mathbf{a}_k[1]) \\
\to & (\mathbf{a}_k[0], & \mathbf{a}_k[1], & \cdots, & \mathbf{a}_i[\gamma-1], & \mathbf{a}_i[\gamma], & 0) \\
\to & \cdots & & & & & & \\
\to & (\mathbf{a}_k[0], & \mathbf{a}_k[1], & \cdots, & \mathbf{a}_k[\gamma-1], & 0, & \mathbf{a}_k[\gamma]) \\
\to & (\mathbf{a}_k[0], & \mathbf{a}_k[1], & \cdots, & \mathbf{a}_k[\gamma-1], & \mathbf{a}_k[\gamma], & 0)
\end{array}
$$

It is easy to verify that the sum of any two adjacent vectors in the above sequence is unique, which gives possible values for $e_{i,j}$'s. Unfortunately, numbers constructed in this way do not necessarily satisfy that $E_i \cap E_{i'} = \emptyset$. In particular, there might exist some pair i', k' with $k' = \tau(i')$ where $\mathbf{a}_i[j] = \mathbf{a}_{i'}[j]$ for $j \leq \gamma - 2$, and $\mathbf{a}_k[j] = \mathbf{a}_{k'}[j]$ for $j \geq \gamma - 1$. In this case, we have

$$(\mathbf{a}_{i'}[0], \mathbf{a}_{i'}[1], \cdots, \mathbf{a}_{k'}[\gamma-1], \mathbf{a}_{k'}[\gamma], 0) = (\mathbf{a}_i[0], \mathbf{a}_i[1], \cdots, \mathbf{a}_k[\gamma-1], \mathbf{a}_k[\gamma], 0),$$

violating $E_i \cap E_{i'} = \emptyset$.

How can we construct unique $e_{i,j}$'s? Towards this, we consider all the one-to-one mappings $g : \mathcal{S}_d^{\gamma+1} \to \mathcal{S}_d^{\gamma+1}$. Under composition of functions, all such mappings form a group $Aut(\mathcal{S}_d^{\gamma+1})$. We are interested in the special mapping that maps each $\mathbf{a}_i$ to $\mathbf{a}_k$ (which corresponds to the permutation τ), which belongs to $Aut(\mathcal{S}_d^{\gamma+1})$. We show that any mapping in $Aut(\mathcal{S}_d^{\gamma+1})$, and hence this special mapping, can be decomposed into a sequence of simple mappings. More precisely, we consider any finite set $\mathcal{M}$ and the group $Aut(\mathcal{M}^\upsilon)$ of one-to-one mappings from $\mathcal{M}^\upsilon$ to itself. We call a mapping in $Aut(\mathcal{M}^\upsilon)$ an h-shuffler if this mapping only changes the h-th coordinate of the input vector, i.e., an h-shuffler f satisfies that for any $\mathbf{y} \in \mathcal{M}^\upsilon$,

$$f(\mathbf{y}) = f(\mathbf{y}[0], \mathbf{y}[1], \cdots, \mathbf{y}[\upsilon-1]) = (\mathbf{y}[0], \cdots, \mathbf{y}[h-1], z, \mathbf{y}[h+1], \cdots, \mathbf{y}[\upsilon-1])),$$

for some $z \in \mathcal{M}$. We show the following group theoretic lemma which states that any mapping of $Aut(\mathcal{M}^\upsilon)$ can be decomposed into 2υ h-shufflers; see Appendix E for the proof. Our $e_{i,j}$'s can be obtained from these h-shufflers.

LEMMA 3.5. *Let $\mathcal{M}$ be a finite set, $\upsilon \in \mathbb{Z}^+$ and $Aut(\mathcal{M}^\upsilon)$ be the group of all one-to-one mappings from $\mathcal{M}^\upsilon$ to itself. The group operation is function composition and denoted as $\circ$. For any $\pi \in Aut(\mathcal{M}^\upsilon)$, there exist h-shufflers $f_h, \hat{f}_h \in Aut(\mathcal{M}^\upsilon)$ for every $1 \le h \le \upsilon$ such that $F(\mathbf{y}) = \hat{F}(\pi(\mathbf{y}))$ for any $\mathbf{y} \in \mathcal{M}^\upsilon$, where $F = f_{\upsilon-1} \circ f_{\upsilon-2} \circ \cdots \circ f_0$ and $\hat{F} = \hat{f}_{\upsilon-1} \circ \hat{f}_{\upsilon-2} \circ \cdots \circ \hat{f}_0$. Furthermore, f_h's and $\hat{f}_h$'s can be constructed in time that is polynomial in $|\mathcal{M}^\upsilon|$.*

Lemma 3.5 implies a decomposition of π into h-shufflers, i.e., $\pi = \hat{f}_0^{-1} \circ \hat{f}_1^{-1} \circ \cdots \circ \hat{f}_{\upsilon-1}^{-1} \circ f_{\upsilon-1} \circ \cdots \circ f_0$.

With Lemma 3.5, we are ready to prove Lemma 3.4. We first set the value of all parameters. Towards this, we will apply Lemma 3.3 twice.

At first, we apply Lemma 3.3 by taking $N = n$. Then we obtain $\sigma : \mathbb{Z}_n \to \mathbb{Z}_{n'}$ where $n' = n^{1+\mathcal{O}(1/\sqrt{\log\log n})}$, $\sigma(i) = \sum_{j=0}^{\gamma} \mathbf{a}_i[j]x^j$ for $\mathbf{a}_i[j] \in \mathcal{S}_d$ where $\gamma = \lceil \frac{\log n}{\log\log n} \rceil + \mathcal{O}(\frac{\log n}{(\log\log n)^{3/2}})$, $d = \lceil e^{(\sqrt{\log\log n}+7)^2} \rceil = e^{\mathcal{O}(\sqrt{\log\log n})} \cdot \log n$ and $x = 5d + 1$. Except N, all the other parameters, including n', σ, x, d, γ and $\mathbf{a}_i$'s are fixed throughout the following part of this section.

Next, we apply Lemma 3.3 again by setting $N = 4\gamma + 4$ where γ takes the value we determined above. By doing so we obtain another injection σ'. We have the following simple observation.

OBSERVATION 1. *If $\ell \le 4\gamma + 4$, then $\sigma'(\ell) = o(\log^2 n) < x^2$.*

Proof. Note that $\gamma = O(\frac{\log n}{\log\log n})$. By Lemma 3.3, $\sigma'(\ell) \le (4\gamma+4)^{1+O(\frac{1}{\sqrt{\log\log(4\gamma+4)}})} = o(\gamma^2) = o(\log^2 n)$. ∎

Next, We will apply Lemma 3.5. In the following part of this paper, any υ-dimensional vector $\mathbf{c}$ represents the number given by the polynomial expression $\sum_{i=1}^{\upsilon-1} \mathbf{c}[i]x^i$; vectors and polynomial expressions are used interchangeably. Given our permutation τ, we define $\hat{\tau}$ as a one-to-one mapping that maps each vector $\mathbf{a}_i$ to $\mathbf{a}_k$, or equivalently, maps $\sigma(i)$ to $\sigma(k)$ if $k = \tau(i)$. Notice that $\mathbf{a}_i$'s form a subset of $\mathcal{S}_d^{\gamma+1}$, so currently $\hat{\tau}$ is only defined on this subset. We can extend $\hat{\tau}$ to $\mathcal{S}_d^{\gamma+1}$ such that for $\mathbf{c} \in \mathcal{S}_d^{\gamma+1}$ and $\mathbf{c} \ne \mathbf{a}_i$, then $\hat{\tau}(\mathbf{c}) = \mathbf{c}$. Hence, $\hat{\tau} \in Aut(\mathcal{S}_d^{\gamma+1})$. According to Lemma 3.5, we can obtain h-shufflers f_h and $\hat{f}_h$ for all $0 \le h \le \gamma$ such that $f_\gamma \circ f_{\gamma-1} \circ \cdots \circ f_0 = \hat{f}_\gamma \circ \hat{f}_{\gamma-1} \circ \cdots \circ \hat{f}_0 \circ \hat{\tau}$.

For ease of notation, define $F_h = f_h \circ f_{h-1} \circ \cdots \circ f_0$ and $\hat{F}_h = \hat{f}_h \circ \hat{f}_{h-1} \circ \cdots \circ \hat{f}_0$. As h-shufflers only changes the h-th coordinate, we have the following observation.

OBSERVATION 2. *The following statements are true:*

- *For any $0 \le h \le \gamma$*
$$(F_\gamma(\mathbf{a}_i))[h] = (F_{\gamma-1}(\mathbf{a}_i))[h] = \cdots = (F_h(\mathbf{a}_i))[h],$$
$$(\hat{F}_\gamma(\mathbf{a}_i))[h] = (\hat{F}_{\gamma-1}(\mathbf{a}_i))[h] = \cdots = (\hat{F}_h(\mathbf{a}_i))[h];$$
- *For any $0 \le h \le \gamma$,*
$$(F_{h-1}(\mathbf{a}_i))[h] = (F_{h-2}(\mathbf{a}_i))[h] = \cdots = (F_0(\mathbf{a}_i))[h] = \mathbf{a}_i[h],$$
$$(\hat{F}_{h-1}(\mathbf{a}_i))[h] = (\hat{F}_{h-2}(\mathbf{a}_i))[h] = \cdots = (\hat{F}_0(\mathbf{a}_i))[h] = \mathbf{a}_i[h].$$

Now we are ready to construct a unique linking sequence for every i. Intuitively, note that since $\sigma'(h) < x^2$, then $\sigma'(h)$ occupies two "bits"in the polynomial (that is, two coordinates). With this in mind, we let $\hat{\sigma}'(i) = (0, \sigma'(i))$. Moreover, we define $\hat{\sigma}'(0) = (0, 0)$. Now consider the following two sequences, starting from $\mathbf{a}_i$ and $\mathbf{a}_k$, respectively, and end up at the same vector:

$$\begin{array}{llllllll}
 & (\mathbf{a}_i[0], & \mathbf{a}_i[1], \cdots, & \mathbf{a}_i[\gamma-1], & \mathbf{a}_i[\gamma], & 0, & \hat{\sigma}'(0)) := & \mathbf{b}_i^0 \\
\to & (0, & \mathbf{a}_i[1], \cdots, & \mathbf{a}_i[\gamma-1], & \mathbf{a}_i[\gamma], & (F_0(\mathbf{a}_i))[0], & \hat{\sigma}'(1)) := & \mathbf{b}_i^1 \\
\to & ((F_0(\mathbf{a}_i))[0], & \mathbf{a}_i[1], \cdots, & \mathbf{a}_i[\gamma-1], & \mathbf{a}_i[\gamma], & 0, & \hat{\sigma}'(2)) := & \mathbf{b}_i^2 \\
\to & ((F_1(\mathbf{a}_i))[0], & 0, \cdots, & \mathbf{a}_i[\gamma-1], & \mathbf{a}_i[\gamma], & (F_1(\mathbf{a}_i))[1], & \hat{\sigma}'(3)) := & \mathbf{b}_i^3 \\
\to & ((F_1(\mathbf{a}_i))[0], & (F_1(\mathbf{a}_i))[1], \cdots, & \mathbf{a}_i[\gamma-1], & \mathbf{a}_i[\gamma], & 0, & \hat{\sigma}'(4)) := & \mathbf{b}_i^4 \\
\to & \cdots & & & & & & \\
\to & ((F_\gamma(\mathbf{a}_i))[0], & (F_\gamma(\mathbf{a}_i))[1], \cdots, & (F_\gamma(\mathbf{a}_i))[\gamma-1], & 0, & (F_\gamma(\mathbf{a}_i))[\gamma], & \hat{\sigma}'(2\gamma+1)) := & \mathbf{b}_i^{2\gamma+1} \\
\to & ((F_\gamma(\mathbf{a}_i))[0], & (F_\gamma(\mathbf{a}_i))[1], \cdots, & (F_\gamma(\mathbf{a}_i))[\gamma-1], & (F_\gamma(\mathbf{a}_i))[\gamma], & 0, & \hat{\sigma}'(2\gamma+2)) := & \mathbf{b}_i^{2\gamma+2}
\end{array}$$

and

$$\begin{array}{lccccccc}
 & (\mathbf{a}_k[0], & \mathbf{a}_k[1], \cdots, & \mathbf{a}_k[\gamma-1], & \mathbf{a}_k[\gamma], & 0, & \hat{\sigma}'(4\gamma+4)) := & \hat{\mathbf{b}}_k^0 \\
\to & (0, & \mathbf{a}_k[1], \cdots, & \mathbf{a}_k[\gamma-1], & \mathbf{a}_k[\gamma], & (\hat{F}_0(\mathbf{a}_k))[0], & \hat{\sigma}'(4\gamma+3)) := & \hat{\mathbf{b}}_k^1 \\
\to & ((\hat{F}_0(\mathbf{a}_k))[0], & \mathbf{a}_k[1], \cdots, & \mathbf{a}_k[\gamma-1], & \mathbf{a}_k[\gamma], & 0, & \hat{\sigma}'(4\gamma+2)) := & \hat{\mathbf{b}}_k^2 \\
\to & ((\hat{F}_1(\mathbf{a}_k))[0], & 0, \cdots, & \mathbf{a}_k[\gamma-1], & \mathbf{a}_k[\gamma], & \hat{F}_1(\mathbf{a}_k))[1]), & \hat{\sigma}'(4\gamma+1)) := & \hat{\mathbf{b}}_k^3 \\
\to & ((\hat{F}_1(\mathbf{a}_k))[0], & \hat{F}_1(\mathbf{a}_k))[1], \cdots, & \mathbf{a}_k[\gamma-1], & \mathbf{a}_k[\gamma], & 0, & \hat{\sigma}'(4\gamma)) := & \hat{\mathbf{b}}_k^4 \\
\to & \cdots & & & & & & \\
\to & ((\hat{F}_\gamma(\mathbf{a}_k))[0], & (\hat{F}_\gamma(\mathbf{a}_k))[1], \cdots, & (\hat{F}_\gamma(\mathbf{a}_k))[\gamma-1], & 0, & (\hat{F}_\gamma(\mathbf{a}_k))[\gamma]), & \hat{\sigma}'(2\gamma+3)) := & \hat{\mathbf{b}}_k^{2\gamma+1} \\
\to & ((\hat{F}_\gamma(\mathbf{a}_k))[0], & (\hat{F}_\gamma(\mathbf{a}_k))[1], \cdots, & (\hat{F}_\gamma(\mathbf{a}_k))[\gamma-1]), & (\hat{F}_\gamma(\mathbf{a}_k))[\gamma], & 0, & \hat{\sigma}'(2\gamma+2)) := & \hat{\mathbf{b}}_k^{2\gamma+2}
\end{array}$$

Consider each vector in the above sequence, say, $\mathbf{b}_i^2$. According to Observation 2, we know

$$\begin{array}{lccccccc}
 & ((F_0(\mathbf{a}_i))[0], & \mathbf{a}_i[1], \cdots, & \mathbf{a}_i[\gamma-1], & \mathbf{a}_i[\gamma], & 0, & \hat{\sigma}'(2)) \\
= & ((F_0(\mathbf{a}_i))[0], & (F_0(\mathbf{a}_i))[1], \cdots, & (F_0(\mathbf{a}_i))[\gamma-1], & (F_0(\mathbf{a}_i))[\gamma], & 0, & \hat{\sigma}'(2))
\end{array}$$

More generally, it is easy to verify that every $\mathbf{b}_i^{2j}$ is the concatenation of the vector $(F_{j-1}(\mathbf{a}_i), 0)$ and $\hat{\sigma}'(2j)$, and each $\mathbf{b}_i^{2j+1}$ is a combination $SW_j\left((F_{j-1}(\mathbf{a}_i), 0)\right)$ and $\hat{\sigma}'(2j+1)$, where SW_j is a one-to-one mapping that swaps two coordinates of a vector. A similar statement holds for $\hat{\mathbf{b}}_k^{2j}$'s and $\hat{\mathbf{b}}_k^{2j+1}$'s. Since SW_j's, F_j's and $\hat{F}_j$'s are all one-to-one mappings, and σ' is an injection, each of the vectors in the sequence above is unique. More precisely, we have the following.

LEMMA 3.6. *For any $0 \le h \le 2\gamma+1$ and $1 \le i \le n$,*

- *If $\mathbf{b}_i^h = \mathbf{b}_{i'}^{h'}$, then $h = h'$ and $i = i'$;*
- *If $\hat{\mathbf{b}}_i^h = \hat{\mathbf{b}}_{i'}^{h'}$, then $h = h'$ and $i = i'$.*

Furthermore, by the fact that $F_\gamma = \hat{F}_\gamma \circ \hat{\tau}$ and $\mathbf{a}_k = \hat{\tau}(\mathbf{a}_i)$, we have the following observation:

OBSERVATION 3.

$$\mathbf{b}_i^{2\gamma+2} = \hat{\mathbf{b}}_k^{2\gamma+2}.$$

Next, we consider any two adjacent vectors in the above sequence. We observe that they differ at exactly three positions – the last coordinate (i.e., $\hat{\sigma}'(j)$'s), and other two coordinates such that one of the two vectors has 0 coordinate. Other coordinates, e.g., $(F_0(\mathbf{a}_i))[0]$ in $\mathbf{b}_i^2$ and $(F_1(\mathbf{a}_i))[0]$ in $\mathbf{b}_i^3$ are identical according to Observation 2. This leads to the following Lemma.

LEMMA 3.7. *For any $0 \le h \le 2\gamma+1$ and $1 \le i \le n$,*

- *If $\mathbf{b}_i^h + \mathbf{b}_i^{h+1} = \mathbf{b}_{i'}^{h'} + \mathbf{b}_{i''}^{h''}$ where $h' \le h''$, then $h' = h$, $h'' = h+1$, $i' = i'' = i$;*
- *If $\hat{\mathbf{b}}_i^h + \hat{\mathbf{b}}_i^{h+1} = \hat{\mathbf{b}}_{i'}^{h'} + \hat{\mathbf{b}}_{i''}^{h''}$ where $h' \le h''$, then $h' = h$, $h'' = h+1$, $i' = i'' = i$.*

Proof. We prove the first statement, that is, $\mathbf{b}_i^h + \mathbf{b}_i^{h+1} = \mathbf{b}_{i'}^{h'} + \mathbf{b}_{i''}^{h''}$ implies $h' = h$, $h'' = h+1$, $i' = i'' = i$. The second statement can be proved in the same way.

We first consider the last coordinate of the summation $\mathbf{b}_i^h + \mathbf{b}_i^{h+1} = \mathbf{b}_{i'}^{h'} + \mathbf{b}_{i''}^{h''}$, which is $\sigma'(h) + \sigma'(h+1) = \sigma'(h') + \sigma'(h'')$. If $h = 0$, according to property 3 of Lemma 3.3, $1 \times \sigma'(1)$ can only be expressed as $0 + \sigma'(1)$, hence we have $h' = 0$ and $h'' = 1$. If $h \ge 1$, according to property 4 of Lemma 3.3, we have $h' = h$ and $h'' = h+1$.

Consider other coordinates of the equation $\mathbf{b}_i^h + \mathbf{b}_i^{h+1} = \mathbf{b}_{i'}^h + \mathbf{b}_{i''}^{h+1}$. On the left-side it is either a or $2a$ where $a \in \mathcal{S}_d$. Consider the equation $a = b + c$ where $b, c \in \{0\} \cup \mathcal{S}_d$. By Lemma 3.2, there do not exist two numbers in $\mathcal{S}$ that add up to a, hence we know $b, c \in \{0, a\}$. Similarly if $2a = b + c$ for $a \in \mathcal{S}_d$ and $b, c \in \{0\} \cup \mathcal{S}_d$, then b and c are both nonzero, for otherwise the linear equation $2y = y_1$ admits a solution, which is a contradiction to that $\mathcal{S}_d$ satisfies Lemma 3.2. Hence, $2a = b + c$ for $a, b, c \in \mathcal{S}_d$, and again by Lemma 3.2 we have $b = c = a$. Hence, we conclude that each coordinate of $\mathbf{b}_{i'}^h$ and $\mathbf{b}_{i''}^{h+1}$ must be the same as $\mathbf{b}_i^h$ and $\mathbf{b}_i^{h+1}$, respectively. By Lemma 3.6, it follows that $i' = i'' = i$. ∎

Using the same argument, we know if $2\mathbf{b}_i^{2\gamma+2} = \mathbf{b}_{i'}^{h'} + \hat{\mathbf{b}}_{i''}^{h''}$, then $\mathbf{b}_i^{2\gamma+2} = \mathbf{b}_{i'}^{h'} = \hat{\mathbf{b}}_{i''}^{h''}$. Thus the following is also true.

LEMMA 3.8. *$2\mathbf{b}_i^{2\gamma+2} = \mathbf{b}_{i'}^{h'} + \hat{\mathbf{b}}_{i''}^{h''}$, then $h' = h'' = 2\gamma + 2$, $(i', i'') = (i, \tau(i))$.*

We are now ready to prove Lemma 3.4.

Proof. [Proof of Lemma 3.4]The sequence of E_i that links $\sigma(i)$ and $\sigma(k)$ for $k = \tau(i)$ is exactly $\mathbf{b}_i^0 = \mathbf{a}_i, \mathbf{b}_i^1, \mathbf{b}_i^2, \cdots, \mathbf{b}_i^{2\gamma+2} = \hat{\mathbf{b}}_k^{2\gamma+2}, \hat{\mathbf{b}}_k^{2\gamma+1}, \cdots, \hat{\mathbf{b}}_k^1, \hat{\mathbf{b}}_k^0 = \mathbf{a}_k$ (recall that by a vector $\mathbf{b}$ we mean the integer $\mathbf{b} \cdot \mathbf{x}$ with $x = 5d + 1 = O(\log n)$). The uniqueness of each linking sequence is ensured by Lemma 3.6 and Lemma 3.7. The largest number is bounded by $x^{\gamma+5} = n^{1+O(\frac{1}{\sqrt{\log\log n}})}$. Furthermore, $\omega = O(\gamma) = O(\frac{\log n}{\log\log n})$. Thus, all properties of Lemma 3.4 are satisfied. ∎

4 Conclusion

In this article we consider the problem $P||\sum_i C_i^q$, that is, identical machine scheduling with the objective of minimizing the ℓ_q-norm of machine loads. We establish a PTAS with running time $2^{\tilde{O}(\sqrt{1/\epsilon})} + n^{O(1)}$ and prove that it is essentially best-possible under the exponential time hypothesis. To the best of our knowledge, this is the first PTAS that runs in sub-exponential time in $1/\epsilon$ for a strongly NP-hard scheduling problem. It is interesting to explore this sub-exponential phenomenon in other settings. In particular, it is a meaningful open problem to consider the classical scheduling problem of minimizing the weighted sum of completion times on parallel machines scheduling, $P||\sum_j w_j C_j$, and determine if it admits a PTAS with subexponential dependency on $1/\epsilon$. We also leave as an open problem open problem to show a subexponential lower bound or to develop an FPTAS for $P||\sum_i C_i^q$ where $m = \Theta((1/\epsilon)^\theta)$ for $\theta \in (1/2, 1]$.

5 Acknowledgments

We thank anonymous reviewers for their constructive comments. The first author was partially funded by the National Science Foundation under grant No. 2004096. The second author was partially funded by the "New Generation of AI 2030" major project (2018AAA0100902). The third author was partially funded by Fondecyt Project Nr. 1181527 and ANID – Millennium Science Initiative Program – NCN17 059.

A Omitted Proofs in Section 2 - Algorithms $\mathbf{AL_1}$, $\mathbf{AL_2}$, $\mathbf{AL_3}$

A.1 Algorithm 1

LEMMA A.1. *Consider an instance after the preprocessing of Lemma 2.1. For any $\epsilon > 0$, there exists an algorithm AL_1 that returns an $(1 + O(\epsilon))$-approximate solution for $P||\sum_i C_i^q$ and runs in time $(m/\epsilon)^{O(m)}$.*

The algorithm can be formulated as a dynamic program. For each $h \in \mathbb{Z}_n$, we create a set of states $\mathcal{F}_h$. A state $(h, L_1, \ldots, L_m)$ belongs to $\mathcal{F}_h$ if it is possible to assign jobs $1, \ldots, h$ on machines such that the total load on machine i equals L_i for all $i \in \mathbb{Z}_m$. Starting from $(0, \ldots, 0) \in \mathcal{F}_0$, every state in $\mathcal{F}_{i-1}$ can give rise to some states in $\mathcal{F}_i$ by trying all the possible assignment of job h. And the solution is given by the state in $\mathcal{F}_n$ with the minimum objective i.e., $\sum_{i=1}^m L_i^q$. Due to Lemma 2.1, we know that one of the optimal solutions has a load vector $(L_1^*, \ldots, L_m^*) \in \mathcal{F}_n$ satisfying $L_i^* \leq 2$ for all i. Clearly, the running time of Algorithm 1 is $O(m \sum_i |\mathcal{F}_i|)$. However, the total number of states stored during the dynamic programming can be pseudo-polynomial. Fortunately, using the framework by Woeginger [41], we are able to trim the state space to make it polynomial. More precisely, for any $\delta > 0$ construct $\hat{\mathcal{F}}_1, \ldots, \hat{\mathcal{F}}_n$ satisfying the following properties:

- the size of each $\hat{\mathcal{F}}_i$ is bounded by $(1/\delta)^{O(m)}$;
- for each $(i, L_1, \ldots, L_m) \in \mathcal{F}_i$, there exists $(i, \hat{L}_1, \ldots, \hat{L}_m) \in \hat{\mathcal{F}}_i$ such that $L_j \leq \hat{L}_j \leq L_j(1 + \delta)^i$.

In Algorithm 1, we set the parameter $\delta = \epsilon/n$. Due to Lemma 2.1, $n \in O(m/\epsilon)$, the total running time is bounded by $(m/\epsilon)^{O(m)}$. Let $(n, \hat{L}_1, \ldots, \hat{L}_m)$ be the state with the minimum objective in $\hat{\mathcal{F}}_n$, and $(n, L_1^*, \ldots, L_m^*)$ be the state with the minimum objective in $\mathcal{F}_n$. Hence, $(n, \hat{L}_1, \ldots, \hat{L}_m)$ is an $(1 + O(\epsilon))$-approximate solution, as,

$$\sum_i \hat{L}_i^q \leq (1 + \delta)^{nq} \sum_i L_i^{*q} \leq (1 + 2q\epsilon)OPT,$$

where q is a fixed constant.

Algorithm 1 Pseudo Code Description of AL_1

Input: I, ϵ
Output: $\min\{\sum_{i=1}^m \hat{L}_i^q : (\hat{L}_1, \ldots, \hat{L}_m) \in \hat{\mathcal{F}}_n\}$

```
δ = ε/n
Γ = {[0], [ε, ε(1+δ)), [ε(1+δ), ε(1+δ)^2), ...}
F̂_0 = {(0, ..., 0)}
for j = 1 to n do
    F'_j = ∅
    for all (L_1, ..., L_m) ∈ F_{i-1} do
        for all i ∈ [1, m] and L_i + p_j ≤ 2 do
            F'_j = F'_j ∪ (L_1, ..., L_{i-1}, L_i + p_j, L_{i+1}, ..., L_m)
        end for
    end for
    F̂_j = ∅
    for all S ∈ Γ^m do
        for i = 1 to m do
            L̂_i = max{L_i : (..., L_i, ...) ∈ F' ∩ S}
        end for
        F̂_j = F̂_j ∪ (L̂_1, ..., L̂_m)
    end for
end for
```

A.2 Algorithm 2

We first recall Lemma 2.2 and then present its proof.

LEMMA 2.2 1. *Consider an instance after the preprocessing of Lemma 2.1. For any $\epsilon > 0$, there exists an algorithm AL_2 that outputs an $(1 + O(\epsilon))$-approximation solution for $P||\sum_i C_i^q$ within $m^{\tilde{O}(1/\sqrt{\epsilon})}$ time.*

Proof. Based on Lemma 2.3, we design AL_2 as a dynamic program as follows.

We say that a vector $(i, v, u_1, u_2, \ldots, u_\tau)$ is a valid state if it is possible to assign the u_h largest jobs in $\mathcal{G}_h$, for each h, on the first i machines with the objective equal to v. In our algorithm, for each $i \in \{0, \ldots, m\}$, we construct a set $\mathcal{F}_i$ of valid states. Start from $(0, \ldots, 0) \in \mathcal{F}_0$. To construct the valid states in $\mathcal{F}_i$, we consider a state in $\mathcal{F}_{i-1}$ and try all possible assignments of jobs to machine i that respect the ordering of jobs given by Lemma 2.3 and the load bound for each machine implied by Lemma 2.1. Given the sets $\mathcal{F}_0, \ldots, \mathcal{F}_m$, the answer can be found by searching the state with the minimum objective in $\mathcal{F}_m$. In order to limit the number of states, we can eliminate *dominated* states. Namely, if two states $(i, v, u_1, \ldots, u_\tau)$ and $(i, v', u_1, \ldots, u_\tau)$ in $\mathcal{F}_i$ satisfy $v' < v$, then we say that $(i, v', u_1, \ldots, u_\tau)$ is dominated and delete it from $\mathcal{F}_i$.

The overall running time of AL_2 can be bounded as follows. Recall that by Lemma 2.1, $p_j \geq \epsilon$ and that the load of each machine is at most 2. Hence, each state in $\mathcal{F}_{i-1}$ can give rise to at most $(2/\epsilon + 1)^\tau$ (dominated or undominated) states in $\mathcal{F}_i$. As also the number of jobs in each set $\mathcal{G}_h$ is bounded by $2m/\epsilon$, the number of undominated states in $\mathcal{F}_i$ is at most $(2m/\epsilon + 1)^\tau = m^{\tilde{O}(\sqrt{1/\epsilon})}$. Hence, each set $\mathcal{F}_i$ can be constructed in time $(2/\epsilon + 1)^\tau m^{\tilde{O}(\sqrt{1/\epsilon})} = m^{\tilde{O}(\sqrt{1/\epsilon})}$, which implies the same bound for the overall running of our dynamic programming algorithm. The lemma follows. ∎

A.3 Algorithm 3

The goal of this subsection is to prove the following lemma.

LEMMA A.2. *Consider an instance after the preprocessing of Lemma 2.1. For any $\epsilon > 0$, there exists an algorithm that outputs a feasible schedule for well structured instance of $P||\sum_i C_i^q$ whose objective value is at most $OPT + O(\log^2 m)$ within $(1/\epsilon)^{O(1)} + n^{O(1)}$ time.*

Recall Lemma 2.1, and that an approximation scheme for well structured instances also implies an approximation scheme for general instances. Given Lemma A.2 and the fact that $OPT \geq m$ for well structured instances, we know that the additive error $O(\log^2 m) \leq \epsilon OPT$ if $m = \Omega(1/\epsilon \log^2(1/\epsilon))$. Hence, Theorem 2.1 is proved.

In the following, we present Algorithm 3, which is the algorithm claimed in Lemma A.2. Algorithm 3 modifies upon the famous algorithm for bin packing by Karmarkar and Karp [23]. Given an instance I, $m(I)$ denotes the number of machines, and size(I) denotes the total processing time of jobs. To exclude the trivial cases, if I consists of less than $m(I)$ jobs, then it is obvious that the optimal solution assigns each job to a separate machine. Hence, in the following parts, we can assume without of loss generality there are more than $m(I)$ jobs.

We give a very high-level description. The scheduling problem can be interpreted as a bin packing problem where the bin number is a constraint, and the objective is to minimize the cost of bins instead of minimizing the number of bins (where the cost of a bin is its load to the power of q). Under such an interpretation, we are able to iteratively apply the *harmonic grouping scheme* to round job processing times and establish a *configuration LP* for the rounded instance. Based on the extreme point solution of the configuration LP, we assign jobs to roughly $m(I)/2$ machines and continue with the remaining jobs and machines.

Algorithm 3 Pseudo Code Description of AL_3

```
I_D = ∅
while m(I) > 10 do
    apply harmonic grouping scheme to create instance I' and I_d
    I_D = I_D ∪ I_d
    let x̂(I') be the (ε/log m)-balanced solution for Conf-IP(I')
    schedule I' \ I'_res(x̂) according to ⌊x̂(I')⌋
    I = I'_res(x̂)
end while
schedule I using Algorithm 1
while ∃i, j s.t. L_i − L_j > 1 do
    move the job with the largest processing time in machine i to machine j
end while
schedule I_D on arbitrarily O(log^2 m) machines with max increasing load 2
```

As discussed in Lemma 2.1, here we assume the processing time of all jobs in I are within $[\epsilon, 1]$. Suppose there are χ distinct job processing times with b_1 jobs of processing time p_1, b_2 jobs of processing time b_2, ... , b_χ jobs of processing time p_χ. Consider the the subset of jobs that can be scheduled on a single machine, which can be characterized by a χ-tuple $\mathbf{t}_j = (t_{1j}, t_{2j}, \cdots, t_{\chi j})$ where t_{ij} indicates the number of jobs of processing time p_i on this machine. We call any $\mathbf{t}_j$ with $t_{ij} \leq b_i$ for every i as a configuration. Let N denote the number of configurations, let $\mathbf{t}_1, \mathbf{t}_2, \cdots, \mathbf{t}_N$ be a complete enumeration of them.

We establish a configuration integer program for instance I as follows. We introduce a variable x_j for each configuration $\mathbf{t}_j$ which indicates the number of machines which is scheduled according to $\mathbf{t}_j$. Consequently, all machines of configuration $\mathbf{t}_j$ accommodate $t_{ij}x_j$ jobs of processing time p_i. We define the load, or total processing time of configuration $\mathbf{t}_j$ as $L(\mathbf{t}_j) = \sum_{i=1}^{\chi} t_{ij}p_i$. We define the cost of configuration $\mathbf{t}_j$ as $v_j = (\sum_{i=1}^{\chi} t_{ij}p_i)^q$.

$$\text{Conf-IP}(I): \quad \min \quad \sum_{j=1}^{N} v_j x_j$$

$$\text{(A.1a)} \qquad s.t. \quad \sum_{j=1}^{N} x_j = m(I)$$

$$\text{(A.1b)} \qquad \sum_{j=1}^{N} t_{ij} x_j \geq b_i, \quad i = 1, 2, \cdots, \chi$$

$$x_j \in \mathbb{N}, \quad j = 1, 2, \cdots, N$$

Let $OPT_{IP}(I)$ be the optimal objective value of Conf-IP(I). Relaxing the integral constraint $x_j \in \mathbb{N}$ to $x_j \geq 0$ in Conf-LP(I), we obtain a configuration linear programming Conf-LP(I). Let $OPT_{LP}(I)$ be its optimal objective value, it is obvious that $OPT_{LP}(I) \leq OPT_{IP}(I)$. We have the following lemma.

LEMMA A.3. *There exists an algorithm of running time polynomial in $\chi, \log(m/\epsilon)$ and $1/\varsigma$, and returns a feasible extreme point solution of objective value at most $OPT_{LP}(I) + \varsigma$ for Conf-LP(I).*

Proof. We consider the dual of Conf-LP(I):

$$\text{Dual-LP}(I): \quad \max \quad \sum_{i=1}^{\chi} b_i y_i - m(I)z$$

$$\text{(A.2a)} \qquad s.t. \quad \sum_{i=1}^{\chi} t_{ij} y_i - z \leq v_j, \quad j = 1, 2, \cdots, N$$

$$y_i \geq 0, \quad j = 1, 2, \cdots, N$$

We use a similar algorithm as that for the classical bin packing problem. We employ the ellipsoid method to solve Dual-LP($\bar{I}$). We give a brief description. The ellipsoid method iteratively computes a sequence of ellipsoids $E_0, E_1, \cdots,$. In each iteration, it implements a separation oracle to check whether the center of the current ellipsoid E_k, say, $(\mathbf{y}^k, z^k) = (y_1^k, \cdots, y_\chi^k, z^k)$, is feasible. If it is, then it outputs a cut $\mathbf{b} \cdot \mathbf{y} - m(I)z \geq \mathbf{b} \cdot \mathbf{y}^k - m(I)z^k$ where $\mathbf{b} = (b_1, \cdots, b_\chi)$; Otherwise, it finds out a violating constraint, say, $\mathbf{t}_j \cdot \mathbf{y} - z \leq \bar{v}_j$, and outputs $\mathbf{t}_j \cdot \mathbf{y} - z \leq \mathbf{t}_j \cdot \mathbf{y}^k - z^k$. Incorporating the cut output by the separation oracle, the ellipsoid method computes a new ellipsoid E_{k+1} and guarantees that the volume of the new ellipsoid is smaller than E_k by a factor of $e^{-\frac{1}{5(\chi+1)}}$. After a polynomial number of iterations (specifically, which is polynomial in $\log 1/\varsigma$), the ellipsoid method finds a near-optimal feasible solution with an additive error of ς.

In their seminal work, Karmarkar and Karp [23] further prove that to compute an approximate solution to Conf-LP(I) up to an additive precision of ς, it suffices to construct an approximate separation oracle such that in each iteration, instead of checking whether the center $(\mathbf{y}^k, z^k)$ is feasible and returns a violating constraint if it is infeasible, the approximate separation oracle checks a point $(\tilde{\mathbf{y}}^k, \tilde{z}^k)$ and does the following:

- If $(\tilde{\mathbf{y}}^k, \tilde{z}^k)$ violates a constraint, say, $\mathbf{t}_j \cdot \tilde{\mathbf{y}}^k - \tilde{z}^k > v_j$, then outputs cut $\mathbf{t}_j \cdot \mathbf{y} - z \leq \mathbf{t}_j \cdot \mathbf{y}^k - \tilde{z}^k$;

- If $(\tilde{\mathbf{y}}^k, \tilde{z}^k)$ does not violate any constraint, then outputs $\mathbf{b}\cdot\mathbf{y} - m(I)z \geq \mathbf{b}\cdot\tilde{\mathbf{y}}^k - m(I)\tilde{z}^k$.

Karmarkar and Karp showed that ellipsoid method equipped with the approximate separation oracle can return a near-optimal solution within an additive error of $O(\varsigma)$ as long as the point $(\tilde{\mathbf{y}}^k, \tilde{z}^k)$ in each iteration satisfies that

- For any constraint, if $\mathbf{t}_j \cdot \tilde{\mathbf{y}}^k - \tilde{z}^k > v_j$, then $\mathbf{t}_j \cdot \mathbf{y}^k - z^k > v_j$;
- For the objective function, $\mathbf{b}\cdot\mathbf{y}^k - m(I)z^k \leq \mathbf{b}\cdot\tilde{\mathbf{y}}^k - m(I)\tilde{z}^k + \varsigma$.

Here the first property ensures that if $(\tilde{\mathbf{y}}^k, \tilde{z}^k)$ is infeasible, then $(\mathbf{y}^k, z^k)$ is also infeasible by violating the same constraint, and therefore the approximate separation oracle proceeds exactly the same as an (accurate) separation oracle. The second property ensures that the approximate separation oracle will never cut off a feasible point whose objective value is significantly better than $(\tilde{\mathbf{y}}^k, \tilde{z}^k)$ by ς, and hence ensures the near-optimality.

Now we describe our approximate separation oracle as follows. We first round v_j *up* to be the nearest value of the form $(1+\epsilon)^k$ and let it be $\bar{v}_j$. Notice that there are a polynomial number of distinct rounded values. Given an arbitrary point $(\mathbf{y}^k, z^k)$, we consider all inequalities of the form

$$\mathbf{t}_j\mathbf{y}^k \leq \bar{v}_j + z^k.$$

Our goal is to find out a violating constraint or determine there is none. Since there are only a polynomial number of different values for $\bar{v}_j$'s, we can sequentially check for every value $(1+\epsilon)^h$, whether $(\mathbf{y}^k, z^k)$ violates the constraint $\mathbf{t}_j\mathbf{y}^k \leq (1+\epsilon)^h + z^k$ for all configurations $\mathbf{t}_j$ such that $(1+\epsilon)^{h-1} < (\mathbf{t}_j\mathbf{p})^q \leq (1+\epsilon)^h$ where $\mathbf{p} = (p_1, \cdots, p_\chi)$. We argue that we can drop the lower bound by sequentially checking for every value $(1+\epsilon)^h$, whether $(\mathbf{y}^k, z^k)$ violates the constraint $\mathbf{t}_j\mathbf{y}^k \leq (1+\epsilon)^h + z^k$ for all configurations $\mathbf{t}_j$ such that $(\mathbf{t}_j\mathbf{p})^q \leq (1+\epsilon)^h$. This is because that if $(\mathbf{y}^k, z^k)$ violates $\mathbf{t}_j\mathbf{y}^k \leq (1+\epsilon)^h + z^k$ but $(\mathbf{t}_j\mathbf{p})^q \leq (1+\epsilon)^{h-1}$, say, $(\mathbf{t}_j\mathbf{p})^q$ rounded up to $(1+\epsilon)^{h'}$ for $h' < h$, then $(\mathbf{y}^k, z^k)$ also violates $\mathbf{t}_j\mathbf{y}^k \leq (1+\epsilon)^{h'} + z^k$, which will be found out already. Hence, finding a violating constraint is equivalent as finding a vector $\mathbf{t} \leq \mathbf{b}$ such that $(\mathbf{t}\cdot\mathbf{p})^q \leq (1+\epsilon)^h$ and $\mathbf{t}\cdot\mathbf{y}^k$ is maximized, and comparing this maximal value with $z^k + (1+\epsilon)^k$. This is a knapsack problem which admits a fully polynomial time approximation scheme (FPTAS). More precisely, using the same method as Karmarkar and Karp [23], we can round $\mathbf{y}^k$ to some value $\tilde{\mathbf{y}}^k$ close enough such that

- For any χ-dimensional vector $\mathbf{d}$ whose coordinates are non-negative and $\|\mathbf{d}\|_1 \leq n$, $0 \leq \mathbf{d}\cdot\mathbf{y}^k - \mathbf{d}\cdot\tilde{\mathbf{y}}^k \leq \varsigma$;
- In polynomial time (specifically, polynomial in $1/\varsigma$), we are able to find $\mathbf{t}^*$ such that by taking $\mathbf{t} = \mathbf{t}^*$, $\mathbf{t}\cdot\tilde{\mathbf{y}}^k$ is maximized subject to $(\mathbf{t}\cdot\mathbf{p})^q \leq (1+\epsilon)^h$.

Overall, our above argument ensures that in polynomial time we either determine some configuration $\mathbf{t}_j$ such that $\mathbf{t}_j\tilde{\mathbf{y}}^k > \bar{v}_j + z^k$ for some j, and hence

$$\mathbf{t}_j\mathbf{y}^k + z^k \geq \mathbf{t}_j\tilde{\mathbf{y}}^k\bar{v}_j + z^k > \bar{v}_j + z^k \geq v_j + z^k;$$

or we conclude there is no such configuration and guarantee that $\mathbf{b}\cdot\mathbf{y}^k - m(I)z^k \leq \mathbf{b}\cdot\tilde{\mathbf{y}}^k - m(I)\tilde{z}^k + \varsigma$.

Hence, there exists an approximate separation oracle for Dual-LP(I), indicating that Dual-LP(I) can be solved using the ellipsoid method in polynomial time (up to arbitrary precision). Notice that the derived solution may not necessarily be an extreme point, however, by using exactly the same argument as that of Karmarkar and Karp [23], we can make it into an extreme point solution. ∎

Consider the near-optimal extreme point solution $\mathbf{x}(I)$ given by Lemma A.3. We denote by $\lfloor\mathbf{x}(I)\rfloor = (\lfloor x_1(I)\rfloor, \cdots, \lfloor x_N(I)\rfloor)$ the rounded solution. Similar as the algorithm for bin packing, we assign jobs to machine according to $\lfloor\mathbf{x}(I)\rfloor$ and then proceed with the residue instance $I_{res}(\mathbf{x})$. Denote by $I_{res}(\mathbf{x})$ the *residue instance* where we take away jobs scheduled according to $\lfloor\mathbf{x}(I)\rfloor$ from the original instance I, i.e., I_{res} consists jobs in $\mathbf{x}(I) - \lfloor\mathbf{x}(I)\rfloor$. It is easy to see $I_{res}(\mathbf{x})$ consists of $m(I_{res}(\mathbf{x})) = m(I) - \sum_j \lfloor x_j(I)\rfloor$ machines.

LEMMA A.4. $OPT_{LP}(I_{res}(\mathbf{x})) + OPT_{LP}(I \setminus I_{res}(\mathbf{x})) \leq OPT_{LP}(I)$

Proof. We know each configuration of instance I is also a configuration of instance I_{res} and $I \setminus I_{res}(\mathbf{x})$. Hence, for any solution $\mathbf{x}(I)$ of instance I, $\lfloor\mathbf{x}(I)\rfloor$ is an feasible solution of $I \setminus I_{res}(\mathbf{x})$ and $\mathbf{x}(I) - \lfloor\mathbf{x}(I)\rfloor$ is an feasible solution of $I_{res}(\mathbf{x})$. ∎

Similar to the bin packing algorithm by Karmarkar and Karp [23], we will employ the *harmonic grouping scheme* to round the instance and apply Lemma A.3 on the rounded instance. It has to be noticed that the processing time of every job in the instance is no more than 1 which is guaranteed by Lemma 2.1. The harmonic grouping works as follows: We deal with the job one by one in non-decreasing order of its processing time and pack the job into the current group. At any time, only one group is open. When the total processing time of jobs in the current group is at least 2, we close it and start a new group. By doing this, all jobs in I are packed into r groups i.e., $G_1, \ldots, G_r$. We discard all jobs in G_1 along with $|G_{i-1}| - |G_i|$ jobs with smallest processing time in G_i for each $i \in [2, r]$ and let it be I_d. For those remaining jobs, we lift the processing time to the largest one among their group and let it be I'. The *harmonic grouping scheme* has the following properties [40]:

- the number of distinct job processing times in I' is at most $\text{size}(I)/2$;
- $\text{size}(I_d) \in O(\log \text{size}(I))$.

LEMMA A.5. $OPT_{LP}(I') \leq OPT_{LP}(I)$

Proof. Given any solution $\mathbf{x}(I)$ of instance I, for each configuration we can replace each job j in group G_i with a job j' in group G_{i+1}. In such a modified solution, all jobs of I' are scheduled. Since $p_{j'} \leq p_j$ holds, the objective does not increase. ∎

Now we formally present Algorithm 3. Given an instance I of the scheduling problem, we first apply the harmonic grouping scheme to obtain the rounded instance I' along with another instance I_d composed by the discarded jobs. Then we apply Lemma A.3 to derive a feasible solution $\mathbf{x}(I')$ for I' and assign jobs to machines according to $\lfloor \mathbf{x}(I') \rfloor$ and close these machines. The remaining jobs $\mathbf{x}(I') - \lfloor \mathbf{x}(I') \rfloor$ and the remaining empty machines $m(I') - \|\lfloor \mathbf{x}(I'_j) \rfloor\|_1$ (here $\|\cdot\|_1$ is the 1-norm, which counts the number of integral configurations) forms a new instance I'_{res}. In the next iteration, I'_{res} servers as the input and we repeat this process until there are only constant machines left. For the instance with constant machines, we call Algorithm 1. Then we do the balancing operation to make sure that for every two machines their load difference is at most 1. At last, we group all the discarded jobs into $O(\log^2 m)$ groups where the total processing time of each group is at most 2. This can be easily done, since the processing time of each job is at most 1. We pick arbitrarily $O(\log^2 m)$ machines and schedule each group of jobs on one machine.

Finally, we estimate the overall loss incurred. In each iteration, only $m(I)/2+1$ variables take non-zero value in solution $I'_{res}(\hat{\mathbf{x}})$. Hence, $m(I'_{res}(\hat{\mathbf{x}})) \leq m(I)/2 + 1$ and after at most $O(\log m)$ rounds Algorithm 3 terminates. Each iteration introduces an additional cost of ς, together with discarded jobs of total processing time $O(\log^2 m)$, which need to be handled at last. Set $\varsigma = O(\log m)$ and observing that $OPT \geq OPT_{LP}(I)$ and $OPT \geq m$, we know the overall additional cost is bounded by $O(\log^2 m)$. Now consider all the discarded jobs. Through Lemma 2.1 and the balancing operation, we know the load of machines in the solution is at most 3. Meanwhile, due to the convexity of the objective, the balancing operation does not increase the objective value. Given that q is a constant, hence, the overall objective value increases by at most $O(\log^2 m)$, and Lemma A.2 is proved.

B Omitted Proofs in Section 3.1 - Proof of Lemma 3.1

The goal of this section is to prove the following lemma.

LEMMA 3.1 1. *Assuming ETH, there exists a constant $\beta \in (0,1)$ such that for any sufficiently small $\epsilon', \delta > 0$, it is not possible to distinguish between instances of 3SAT′ with $(1-\epsilon') \cdot 4n/3$ clauses where at least $4n/3$ clauses are satisfiable, from instances where at most $(\beta + \epsilon') \cdot 4n/3$ clauses are satisfiable, in time $2^{O(n^{1-\delta})}$.*

Towards the proof, we start with the following result (see, e.g., Corollary 1 of [5]).

LEMMA B.1. *[34, 5] Under ETH, for sufficiently small $\epsilon' > 0$, and $\delta > 0$, it is impossible to distinguish between instances of 3SAT with Λ clauses where at least $(1-\epsilon')\Lambda$ are satisfiable from instances where at most $(7/8+\epsilon')\Lambda$ are satisfiable, in time $O(2^{\Lambda^{1-\delta}})$.*

Applying the classical technique of constructing enforcer via expander for 3SAT (see, e.g., Theorem 5 of [39]), we have the following,

LEMMA B.2. *[39] There exists a constant d_0 such that given a 3SAT formula ϕ with Λ clauses, another 3SAT formula ϕ' with $\Lambda' = \Lambda + 3d_0\Lambda = O(\Lambda)$ clauses can be constructed in polynomial time such that:*

- *Every variable occurs in at most $2d+1$ clauses in ϕ';*
- *There is an assignment for ϕ where at most k clauses are not satisfied if and only if there is an assignment for ϕ' such that at most k clauses are not satisfied.*

Denote by 3SAT-d the 3SAT problem where every variable occurs at most d times. Combining Lemma B.1 and Lemma B.2, we have the following lemma.

LEMMA B.3. *Under ETH, there exists some constants $d \in \mathbb{N}$ and $\alpha \in (0,1)$ such that for sufficiently small $\epsilon' > 0$, and $\delta > 0$, it is impossible to distinguish between instances of 3SAT-d with Λ clauses where at least $(1-\epsilon')\Lambda$ are satisfiable from instances where at most $(\alpha+\epsilon')\Lambda$ are satisfiable, in time $O(2^{\Lambda^{1-\delta}})$.*

It is worth mentioning that the reduction in [39] involves constructing 2-clauses, that is, 3SAT-d in Lemma B.3 refers to a 3SAT instance where clauses may contain 2 or 3 variables. For ease of presentation, we want to enforce every clause to contain exactly 3 variables[1]. This can be done by introducing dummy variables together with 3-clauses that enforce a dummy variable to be true or false (called enforcers). In particular, Berman et al. [4] provide a general enforcer that allows them to deduce the APX-hardness of MAX3SAT (where every clause contains 3 variables and every variable appears 4 times) through the APX-hardness of MAX2SAT. We can apply their technique directly to get a strengthened version of Lemma B.3 where in 3SAT-d every clause contains exactly 3 variables.

The following proof is a slight variation of that from Tovey [38].

LEMMA B.4. *Given a 3SAT-d formula ϕ with Λ clauses, a 3SAT$'$ formula ϕ' with $|C_1| = \Lambda$ and $|C_2| \le 3\Lambda$ clauses can be constructed in polynomial time such that:*

- *If there is an assignment for ϕ where at most k clauses are not satisfied, then there is an assignment for ϕ' where at most k clauses are not satisfied.*
- *If there is an assignment for ϕ' where there are at most k clauses not satisfied, then there is an assignment for ϕ where at most kd clauses are not satisfied.*

Proof. Let z be any variable in ϕ and suppose it appears $\ell \le d$ times in clauses. If $\ell = 1$ then we add a dummy clause $(z \oplus \neg z)$. Otherwise $\ell \ge 2$ and we introduce ℓ new variables z_1, z_2, $\cdots$, z_ℓ and ℓ new clauses $(z_1 \oplus \neg z_2)$, $(z_2 \oplus \neg z_3)$, $\cdots$, $(z_\ell \oplus \neg z_1)$ which enforce $z_1, z_2, \cdots, z_\ell$ to take the same truth value. Meanwhile we replace the ℓ occurrences of z in the original clauses by z_1, z_2, $\cdots$, z_ℓ in turn and remove z. By doing so we transform ϕ into a new formula ϕ' by introducing at most 3Λ new variables and 3Λ new clauses.

Notice that each new clause we add in ϕ' is of the form $(z_i \oplus \neg z_{i'})$. We let C_2 be the set of them and let C_1 be the set of other clauses. It is easy to verify that ϕ' is an instance of 3SAT$'$. Notice that every clause in C_1 has a corresponding clause in ϕ by replacing z_i's with z.

Suppose there is an assignment for ϕ where at most k clauses are not satisfied. Then for any variable z in ϕ that occurs ℓ times, we let z_1, z_2, $\cdots$, z_ℓ all take the same value as z. It is easy to see that at most k clauses in C_1 of ϕ' are not satisfied.

Suppose there is an assignment for ϕ' where at most k clauses in C_1 are not satisfied. For any variable z in ϕ that correspond to z_1, z_2, $\cdots$, z_ℓ in ϕ', we let z_i take the same value of z_1 for all i. Now we check the number of additional unsatisfied clauses in C_1 we introduce by doing so. If all z_i's take the same value, then no additional unsatisfied clauses are introduced. Otherwise, it is possible that some of the clauses in C_1, which is satisfied by z_2, z_3, $\cdots$ or z_ℓ, becomes unsatisfied. But there are at most $d-1$ such kind of clauses. Hence, at most $d-1$ unsatisfied clauses are introduced, if there is at least one unsatisfied clause among $(z_1 \oplus \neg z_2)$, $(z_2 \oplus \neg z_3)$, $\cdots$, $(z_\ell \oplus \neg z_1)$. This implies that we have introduced at most $k(d-1)$ unsatisfied clauses by setting $z_i = z_1$ for all variables, i.e., there are at most $k(d-1)+k = kd$ unsatisfied clauses in C_1 now. Hence, there is an assignment for ϕ where at most kd clauses are not satisfied. ∎

[1] We remark, however, that our reduction also works if C_1 contains 2-causes and 3-clauses. It suffices to create two CL_ℓ, one true copy and one false copy instead of three, and meanwhile adjust the number of dummy jobs.

Given Lemma B.4, we know that if there is an assignment for ϕ with are at most $\alpha\Lambda$ unsatisfied clauses, then there is an assignment for ϕ' with every clause in C_2 satisfied and at most $\alpha\Lambda$ clauses in C_1 unsatisfied; if every assignment has at least $\beta\Lambda$ unsatisfied clauses in ϕ, then there are at least $\beta\Lambda/d$ unsatisfied clauses in ϕ'. Hence, according to Lemma B.3, Lemma 3.1 is proved.

C Omitted Contents in Section 3.2 - Why Old Reduction does not Work

Chen et al. [7] provided a reduction that meets the conditions CO1 to CO4 with job processing times, and hence the target value T, being $O(n^{1+\delta})$ for any arbitrary small constant $\delta > 0$. This reduction provides a strong lower bound for $P||C_{max}$, but does not work well for our problem $P||\sum_i C_i^q$. To see this, we take $q = 2$ as an example and compare the two objective values for the constructed scheduling problem when I_{sat} is satisfiable and when it is not. If I_{sat} is satisfiable, then there exists a schedule such that every machine has a load of exactly T, implying that the optimal objective value is mT^2. Otherwise, at least one machine has load $T+1$ or more and machine has load $T-1$ or less, and then the optimal objective is at least $(m-2)T^2+(T+1)^2+(T-1)^2 = mT^2+2$. For the sake of contradiction, let us assume that there exists a PTAS with running time $2^{O((1/\epsilon)^\kappa)}$ for some $\kappa \in (0,1)$. If we take ϵ to be sufficiently small such that $mT^2\epsilon \leq 1$, then the PTAS can be used to determine whether the constructed scheduling instance admits a feasible schedule of objective value at most $mT^2+1 < mT^2+2$, and hence whether I_{sat} is satisfiable. The running time of the PTAS becomes $2^{O((1/\epsilon)^\kappa)} = 2^{O((mT^2)^\kappa)}$. Plug in $m = O(n)$ and $T = O(n^{1+\delta})$ in the reduction, we have $mT^2 = O(n^{3+2\delta})$. Hence, if $\kappa = 1/3-\delta$, we have $(mT^2)^\kappa \leq O(n^{1-\delta})$, and an efficient PTAS of running time $2^{O((1/\epsilon)^{1/3-\delta})}$ can thus determine the satisfiability of I_{sat} in $2^{O(n^{1-\delta})}$ time, contradicting ETH. To summarize, the above argument implies a lower bound of $2^{O((1/\epsilon)^{1/3-\delta})}$ on the running time of PTAS for arbitrary constant $\delta > 0$, which is not strong enough to match our algorithms in Theorem 2.1.

To overcome the obstacle, a natural idea is to decrease the value of m or T in the reduction. However, if, say, $m = n^{0.9}$ and $T = n^{O(1)}$, then we know the standard dynamic programming for scheduling returns the optimal solution in $T^{O(m)} = 2^{O(n^{0.9})}$ time; similarly, if $T = n^{0.9}$ and $m = n^{O(1)}$, then we know there are at most $n^{0.9}$ different kinds of jobs, and the scheduling problem can also be solved in time $2^{O(n^{0.9})}$ through dynamic programming. Hence, we cannot expect to reduce I_{sat} to such scheduling instances, assuming ETH.

As a consequence, in this paper, we will not try to decrease m or T. Instead, we increase the gap between the two optimal objective values for the constructed scheduling problem when I_{sat} is satisfiable and when I_{sat} is not satisfiable by exploiting the hardness gap in Lemma 3.1.

D Omitted Proofs in Section 3.4 - Proof of Lemma 3.2

LEMMA 3.2 1. *Let $N \in \mathbb{Z}^+$. There exists a subset $\mathcal{S} \subseteq \mathbb{Z}_N$ such that $|\mathcal{S}| \geq N^{1-c_0\sqrt{\frac{1}{\log N}}}$ for some sufficiently large c_0 (in particular, $c_0 = 7$ suffices), and for any $y \in \mathcal{S}$ and $1 \leq h \leq 5$, the linear equation $h \cdot y = y_1 + y_2 + \cdots + y_h$ with $y_i \in \mathcal{S}$ for all i has a unique solution $y_1 = y_2 = \cdots = y_h = y$.*

Proof. For any $d \geq 2$, $M \geq 2$ and $k \leq (M+1)(d-1)^2$, we let $x = 5d-1$ and $\mathbf{x} = (1, x, x^2, \cdots, x^M)$, $\mathbf{c} = (\mathbf{c}[0], \mathbf{c}[1], \cdots, \mathbf{c}[M])$. We define the set $S_k(M,d)$ as:

$$\begin{aligned}\mathcal{S}_k(M,d) \quad = \quad & \{y : y = \mathbf{cx} + x^{M+1} = \mathbf{c}[0] + \mathbf{c}[1]x^1 + \cdots + \mathbf{c}[M]x^M + x^{M+1}, \\ & \mathbf{c}[i] \in \mathbb{N}, 0 \leq \mathbf{c}[i] < d, \sum_{i=0}^{M} (\mathbf{c}[i])^2 = k\}.\end{aligned}$$

That is, $\mathcal{S}_k(M,d)$ is the set of all integers which can be expressed in the form of $\mathbf{c}\cdot\mathbf{x}+x^{M+1}$ such that $\mathbf{c}[i] \in [0,d)$ and $\|\mathbf{c}\|_2^2 = k$, where $\|\cdot\|_2$ is the ℓ_2-norm of a vector.

We claim that for any $y \in \mathcal{S}_k(M,d)$ and $1 \leq h, h' \leq 5$, if $hy = y_1 + y_2 + \cdots + y_{h'}$, $y_i \in S_k(M,d)$, then we have $h' = h$, $y_1 = y_2 = \cdots = y_h = y$. Let $y = \mathbf{cx} + x^{M+1}$ and $y_j = \mathbf{c}_j\mathbf{x} + x^{M+1}$, where $\mathbf{c}_j = (\mathbf{c}_j[0], \mathbf{c}_j[2], \cdots, \mathbf{c}_j[M])$. Using the fact that $x = 5d-1$ and $\mathbf{c}_j[i] < d$, we know for $h' \leq 5$ we have $\sum_{j=1}^{h'} \mathbf{c}_j[i] < x$. Hence by checking the coefficient of x^{M+1} on both sides of the equation $hy = \sum_{j=1}^{h'} y_j$, we have $h = h'$. Moreover, we can conclude that the coefficient of x^i in the sum $y_1 + y_2 + \cdots + y_{h'}$ equals $\sum_{j=1}^{h'} \mathbf{c}_j[i]$, hence by comparing the coefficient of x^i on

both sides, we have

$$\sum_{j=1}^{h} \mathbf{c}_j = h\mathbf{c}.$$

By the definition of $\mathcal{S}_k(M,d)$, the followings are true:

$$\sum_{i=0}^{M} (\mathbf{c}_j[i])^2 = k, \quad \forall j,$$

and

$$\sum_{i=0}^{M} (\frac{\sum_{j=1}^{h} \mathbf{c}_j[i]}{h})^2 = \sum_{i=0}^{M} (\mathbf{c}[i])^2 = k$$

Hence,

$$k = \sum_{i=0}^{M} (\frac{\sum_{j=1}^{h} \mathbf{c}_j[i]}{h})^2 = \frac{\sum_{j=1}^{h} \sum_{i=0}^{M} (\mathbf{c}_j[i])^2}{h} = \sum_{i=0}^{M} \frac{\sum_{j=1}^{h} (\mathbf{c}_j[i])^2}{h}. \tag{D.3}$$

According to the inequality between the quadratic mean and arithmetic mean, we know

$$\left(\frac{\sum_{j=1}^{h} \mathbf{c}_j[i]}{h} \right)^2 \leq \frac{\sum_{j=1}^{h} (\mathbf{c}_j[i])^2}{h}, \quad \forall i$$

and the equality only holds when $c_j[i]$'s are identical for all j. Hence by Eq (D.3) we know $\mathbf{c} = \mathbf{c}_j$, and consequently $y_j = y$ for $1 \leq j \leq h$. Hence, the claim is true.

It remains to select an appropriate $\mathcal{S}_k(M,d)$ such that $\mathcal{S}_k \subseteq \mathbb{Z}_N$ and has a large cardinality. Towards this, we first observe that the largest number of $\mathcal{S}_k(M,d)$ is bounded by $(5d-1)^{M+2}$. We shall select d and M such that $(5d-1)^{M+2} \leq N$. Notice that there are $d^{M+1}-1$ different positive integer numbers which can be expressed as $\mathbf{c} \cdot \mathbf{x} + x^{M+1}$ where $\mathbf{c}[i] \in [0,d)$. Furthermore $k = \|\mathbf{c}\|_2^2 \leq (M+1)(d-1)^2$, hence there exists some $1 \leq k^* \leq (M+1)(d-1)^2$ such that

$$|\mathcal{S}_{k^*}(M,d)| \geq \frac{d^{M+1}-1}{(M+1)(d-1)^2} > \frac{d^{M-1}}{M+1}.$$

It remains to select d and M subject to $(5d-1)^{M+2} \leq N$ such that $\frac{d^{M-1}}{M+1}$ is large. Below all logarithms are taken with the base e. We pick $M = \lfloor \sqrt{\frac{\log N}{\log 5}} \rfloor - 2$ and $d = \lfloor \frac{e^{\sqrt{\log 5 \cdot \log N}}}{5} \rfloor$. It is easy to see that $(M+2)\log(5d-1) \leq \sqrt{\frac{\log N}{\log 5}} \cdot \log e^{\sqrt{\log 5 \cdot \log N}} = \log N$, hence $(5d-1)^{M+2} \leq N$. Furthermore, for sufficiently large N (e.g., $N > e^{10}$), we know $d \geq \frac{e^{\sqrt{\log 5 \cdot \log N}}}{10}$, hence

$$\begin{aligned}
\frac{d^{M-1}}{M+1} = e^{(M-1)\log d - \log(M+1)} &\geq e^{(\sqrt{\log N \cdot \log 5} - \log 10)(\sqrt{\frac{\log N}{\log 5}} - 4) - \frac{1}{2}(\log\log N - \log\log 5)} \\
&\geq e^{\log N \cdot (1 + \frac{-4\sqrt{\log N \log 5} - \frac{\log 10}{\sqrt{\log 5}} \sqrt{\log N} - \Omega(\log\log N)}{\log N})} \\
&= N^{1 - \Omega(\frac{1}{\sqrt{\log N}})}
\end{aligned}$$

Hence, Lemma 3.2 is proved. In particular, it is easy to verify that $4\sqrt{\log 5} + \frac{\log 10}{\sqrt{\log 5}} \leq 7$, hence the $\frac{d^{M-1}}{M+1} \geq N^{1 - \frac{7}{\sqrt{\log N}}}$. ∎

E Omitted proofs in Section 3.4 - Proof of Lemma 3.4

The goal of this subsection is to prove the following.

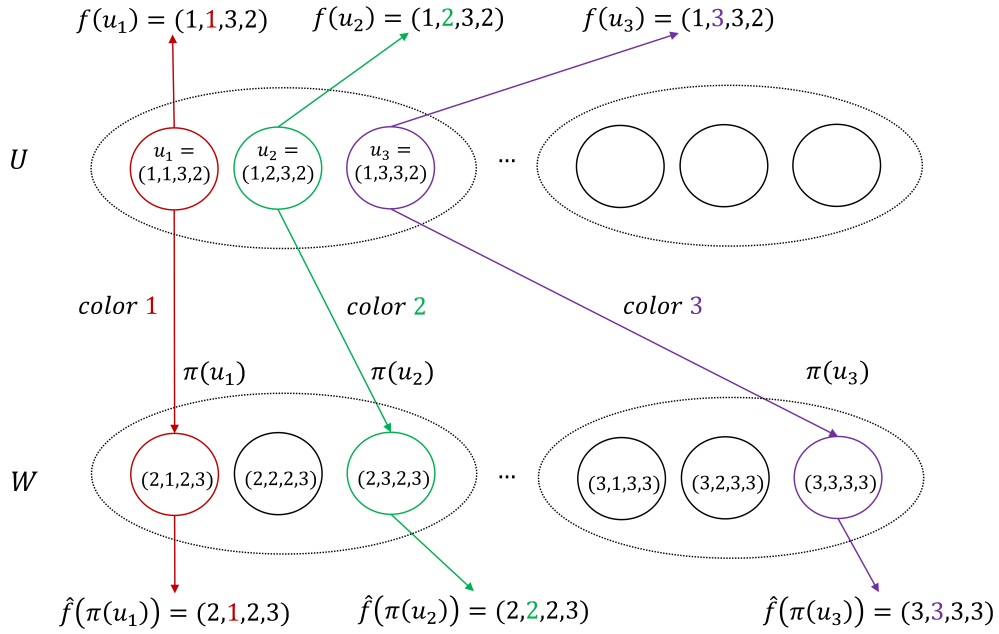

Figure 2: Illustration of Lemma E.1 for $M = \{1, 2, 3\}$, $\upsilon = 4$, $h = 1$ (recall that vectors start with the 0-th coordinate). Each vertex in U (i.e., the solid circle) is a 4-dimensional vector. Each mega-vertex in $\bar{U}$ (i.e., the dotted circle) contains exactly 3 vertices. Each solid line between vertices represents the mapping π which is colored by one of 3 colors.

LEMMA 3.5 1. *For any $\pi \in Aut(\mathcal{M}^\upsilon)$, there exist h-shufflers $f_h, \hat{f}_h \in Aut(\mathcal{M}^\upsilon)$ for every $1 \le h \le \upsilon$ such that $F(\mathbf{y}) = \hat{F}(\pi(\mathbf{y}))$ for any $\mathbf{y} \in \mathcal{M}^\upsilon$, where $F = f_{\upsilon-1} \circ f_{\upsilon-2} \circ \cdots \circ f_0$ and $\hat{F} = \hat{f}_{\upsilon-1} \circ \hat{f}_{\upsilon-2} \circ \cdots \circ \hat{f}_0$. Furthermore, f_h's and $\hat{f}_h$'s can be constructed in time that is polynomial in $|\mathcal{M}^\upsilon|$.*

For any finite set X, we denote by $Aut(X)$ the set of all one-to-one mapping from X to itself. For any $f_1, f_2 \in Aut(X)$, we denote by $f_1 \circ f_2 \in Aut(X)$ the composition of f_1 and f_2, i.e., $f_1 \circ f_2(x) = f_1(f_2(x))$ for any $x \in X$. Note that $Aut(X)$ is a symmetric group under composition. We denote by $f^{-1} \in Aut(X)$ the inverse of $f \in Aut(X)$.

Let $\mathcal{M}$ be an arbitrary finite set of cardinality t. Let $\upsilon \in \mathbb{Z}^+$. Denote by $\mathcal{M}^\upsilon$ the set of all υ-dimensional vectors whose entries belong to $\mathcal{M}$.

For any vector $\mathbf{y} \in \mathcal{M}^\upsilon$, we denote by $\mathbf{y}[h] \in \mathcal{M}$ the h-th coordinate of $\mathbf{y}$, and $\mathbf{y}[-h] \in \mathcal{M}^{\upsilon-1}$ the vector obtained by removing the h-th coordinate from $\mathbf{y}$.

For any $f \in Aut(\mathcal{M}^\upsilon)$ and $0 \le h \le \upsilon - 1$, we call f an h-shuffler if $(f(\mathbf{y}))[-h] = \mathbf{y}[-h]$ for all $y \in \mathcal{M}^\upsilon$, that is,

$$f(\mathbf{y}) = f(\mathbf{y}[0], \mathbf{y}[1], \cdots, \mathbf{y}[\upsilon-1]) = (\mathbf{y}[0], \cdots, \mathbf{y}[h-1], z, \mathbf{y}[h+1], \cdots, \mathbf{y}[\upsilon-1])),$$

for some $z \in \mathcal{M}$.

We prove the following lemma.

LEMMA E.1. *For any $\pi \in Aut(\mathcal{M}^\upsilon)$ and $0 \le h \le \upsilon - 1$, there exist h-shufflers $f, \hat{f} \in Aut(\mathcal{M}^\upsilon)$ such that for any $\mathbf{y} \in \mathcal{M}^\upsilon$, $(f(\mathbf{y}))[h] = \left(\hat{f}(\pi(\mathbf{y}))\right)[h]$. Furthermore, f and $\hat{f}$ can be constructed in time that is polynomial in $|\mathcal{M}^\upsilon|$.*

Briefly speaking, f and $\hat{f}$ shuffles the h-coordinate of $\mathbf{y}$ and its image under π such that they become identical.

Now we are ready to prove Lemma 3.5.

Proof. [Proof of Lemma 3.5] For ease of presentation, we let $\mathcal{M} = \{1, 2, \cdots, t\}$. Note that $|\mathcal{M}^\upsilon| = t^\upsilon$. Sort elements (vectors) of $\mathcal{M}^\upsilon$ in an arbitrary order and denote them by $\{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_{t^\upsilon}\}$. We create a bipartite graph $G = (U \cup W, E)$ to represent π as follows: Both U and W contain t^υ vertices. Let $U = \{u_1, u_2, \cdots, u_{t^\upsilon}\}$ and $W = \{w_1, w_2, \cdots, w_{t^\upsilon}\}$. There is an edge between u_i and w_j if and only if $\pi(\mathbf{c}_i) = \mathbf{c}_j$. Since π is one-to-one mapping, G is 1-regular.

As each u_i and w_i correspond to $\mathbf{c}_i$, we will slightly abuse notation and write $u_i[h]$ or $u_i[-h]$ to refer to $\mathbf{c}_i[h]$ and $\mathbf{c}_i[-h]$.

Contraction. We contract the graph G as follows. We partition U (or W) into $t^{\upsilon-1}$ subsets such that u_i and u_j (or w_i and w_j) are in the same subset if and only if $u_i[-h] = u_j[-h]$ (or $w_i[-h] = w_j[-h]$). Denote by $\bar{U}_i$ (or $\bar{W}_i$), $1 \le i \le t^{\upsilon-1}$, all the subsets in the partition of U (or W). It is clear that each $\bar{U}_i$ (or $\bar{W}_i$) contains exactly t vertices from U (or W). We now contract all the t vertices in $\bar{U}_i$ (or $\bar{W}_i$) into one mega-vertex, and denote this mega-vertex as $\bar{u}_i$ (or $\bar{w}_i$). By doing so we generate parallel edges, that is, there are ℓ parallel edges between each pair of mega-vertices $\bar{u}_i$ and $\bar{w}_j$ if there are ℓ edges between vertices in $\bar{U}_i$ and $\bar{W}_j$ in the original graph G. We denote by ψ an arbitrary one-to-one mapping between a parallel edge (between mega-vertices $\bar{u}_i$ and $\bar{w}_j$) and an edge in G (between some vertex in subset $\bar{U}_i$ and some vertex in subset $\bar{W}_j$). Denote by $\bar{G} = (\bar{U} \cup \bar{W}, \bar{E})$ the contracted graph. Given that G is 1-regular and every mega-vertex contains exactly t vertices, we have the following observation:

OBSERVATION 4. *The contracted graph $\bar{G} = (\bar{U} \cup \bar{W}, \bar{E})$ is a t-regular bipartite graph.*

Coloring. It is known that every bipartite regular graph admits a perfect matching (see, e.g. [26]). Consequently, every t-regular bipartite graph can be decomposed into t perfect matchings. We decompose $\bar{G}$ into t perfect matchings and color edges in each perfect matching with a distinct color. Overall we have used t colors. Since $|\mathcal{M}| = t$, we can map the i-th color to integer $i \in \mathcal{M}$.

Recall that ψ is a one-to-one mapping between $\bar{E}$ and E, hence via ψ we also obtain a coloring for E (by coloring each edge in E with the same color as its corresponding edge in $\bar{E}$). Recall that G is 1-regular. Thus we can extend the edge coloring to a vertex coloring, such that each vertex in G is colored with the same color as the unique edge incident to it.

Define functions f and $\hat{f}$. Consider every vertex set $\bar{U}_k$. We know $\bar{U}_k$ contains t vertices, and let $\bar{U}_k = \{u_{k_1}, u_{k_2}, \cdots, u_{k_t}\}$. By definition $u_{k_i}[-h]$'s are identical and $u_{k_i}[h]$'s are exactly the t elements in $\mathcal{M}$. Recall that we decompose $\bar{G}$ into t perfect matchings and each perfect matching is colored with a unique color, we know the t parallel edges incident to the mega-vertex $\bar{u}_k$ are colored with t distinct colors. Consequently, each vertex u_{k_i} is also colored with a distinct color. Recall the one-to-one correspondence between a vertex u_j and $\mathbf{c}_j \in \mathcal{M}$. Now we define a function f such that $f(\mathbf{c}_{k_i})[-h] = \mathbf{c}_{k_i}[-h]$, and $f(\mathbf{c}_{k_i})[h]$ equals the color of u_{k_i}, where we interpret each color as a number in $\{1, \cdots, t\}$. Consequently, $(f(\mathbf{c}_{k_1}), f(\mathbf{c}_{k_2}), \cdots, f(\mathbf{c}_{k_t}))$ is a permutation of $(\mathbf{c}_{k_1}, \mathbf{c}_{k_2}, \cdots, \mathbf{c}_{k_t})$. Hence, $f \in Aut(\mathcal{M}^\upsilon)$.

Similarly, we consider each $\bar{W}_k = \{w_{k_1}, w_{k_2}, \cdots, w_{k_t}\}$ and define a function $\hat{f}$ such that $\hat{f}(\mathbf{c}_{k_i})[-h] = \mathbf{c}_{k_i}[-h]$, and $\hat{f}(\mathbf{c}_{k_i})[h]$ equals the color of w_{k_i}. Consequently, $(\hat{f}(\mathbf{c}_{k_1}), \hat{f}(\mathbf{c}_{k_2}), \cdots, \hat{f}(\mathbf{c}_{k_t}))$ is also a permutation of $(\mathbf{c}_{k_1}, \mathbf{c}_{k_2}, \cdots, \mathbf{c}_{k_t})$, and $\hat{f} \in Aut(\mathcal{M}^\upsilon)$.

Furthermore, the color of each u_i or w_j is defined as the color of the edge incident to it, hence if there is an edge between u_i and w_j in G, then we know $(f(u_i))[h] = (\hat{f}(w_j))[h]$. Hence, Lemma E.1 is proved. ∎

See Figure 2 for an illustration of the mapping $f, \hat{f}$ we construct in Lemma E.1. Iteratively applying Lemma E.1, we are able to prove the following.

Proof. We prove the following statement by induction: For $0 \le k \le \upsilon - 1$, there exist h-shufflers $f_h, \hat{f}_h \in Aut(\mathcal{M}^\upsilon)$ for every $0 \le h \le k$ such that $F_k(\mathbf{y})[h'] = \hat{F}_k(\pi(\mathbf{y}))[h']$ for any $\mathbf{y} \in \mathcal{M}^\upsilon$ and $h' \le k$, where $F_k = f_k \circ f_{k-1} \circ \cdots \circ f_0$ and $\hat{F}_k = \hat{f}_k \circ \hat{f}_{k-1} \circ \cdots \circ \hat{f}_0$.

The statement is true for $k = 1$ by Lemma E.1. Suppose the statement is true for k, we prove it is true for $k + 1$.

Consider $\hat{f}_k \circ \hat{f}_{k-1} \circ \cdots \circ \hat{f}_0 \circ \pi \circ f_0^{-1} \circ f_1^{-1} \circ \cdots \circ f_k^{-1}$. According to Lemma E.1, there exist $(k+1)$-shufflers $f_{k+1}, \hat{f}_{k+1}$ such that

$$(f_{k+1}(\mathbf{y}))[k+1] = \left(\left(\hat{f}_{k+1} \circ \hat{f}_k \circ \cdots \circ \hat{f}_0 \circ \pi \circ f_0^{-1} \circ f_2^{-1} \circ \cdots \circ f_k^{-1}\right)(\mathbf{y})\right)[k+1], \quad \forall \mathbf{y} \in \mathcal{M}^\upsilon. \tag{E.4}$$

Since $f_k \circ f_{k-1} \circ \cdots \circ f_1 \in Aut(\mathcal{M}^\upsilon)$, for every $\mathbf{y} \in \mathcal{M}^\upsilon$ there exists some $\mathbf{z} \in \mathcal{M}^\upsilon$ such that $\mathbf{y} = (f_k \circ f_{k-1} \circ \cdots \circ f_1)(\mathbf{z})$, plug this into Equation (E.4), for all $\mathbf{z} \in \mathcal{M}^\upsilon$ we get

$$\begin{aligned} & (f_{k+1} \circ f_k \circ \cdots \circ f_0)(\mathbf{z})[k+1] \\ = & \left(\left(\hat{f}_{k+1} \circ \cdots \circ \hat{f}_0 \circ \pi \circ f_0^{-1} \circ \cdots \circ f_k^{-1} \circ f_k \circ f_{k-1} \circ \cdots \circ f_0\right)(\mathbf{z})\right)[k+1] \\ = & \left(\left(\hat{f}_{k+1} \circ \cdots \circ \hat{f}_0 \circ \pi\right)(\mathbf{z})\right)[k+1] \end{aligned}$$

Moreover, for any $h \le k$, recall that f_{k+1} and $\hat{f}_{k+1}$ does not change the h-th coordinate, hence

$$\begin{aligned} (f_{k+1} \circ f_k \circ \cdots \circ f_0)(\mathbf{z})[h] &= (f_k \circ \cdots \circ f_0)(\mathbf{z})[h] \\ &= \left(\left(\hat{f}_k \circ \cdots \circ \hat{f}_1 \circ \pi\right)(\mathbf{z})\right)[h] \\ &= \left(\left(\hat{f}_{k+1} \circ \hat{f}_k \circ \cdots \circ \hat{f}_0 \circ \pi\right)(\mathbf{z})\right)[h] \end{aligned}$$

Hence, the statement holds for all $k \le \upsilon - 1$, and Lemma 3.5 is proved. ∎

F Construction of the Scheduling Instance

Now we provide the details of the reduction. We first recall all the functions and parameters we have set in proving Lemma 3.4.

- Recall that τ is the one-to-one mapping that maps i to k for every $(z_i \oplus \neg z_k) \in C_2$.
- Apply Lemma 3.3 by taking $N = n$, we get $\sigma : \mathbb{Z}_n \to \mathbb{Z}_{n'}$ where $n' = n^{1+\mathcal{O}(\frac{1}{\sqrt{\log\log n}})}$, $\sigma(i) = \sum_{j=0}^{\gamma} \mathbf{a}_i[j]x^j$ for $\mathbf{a}_i[j] \in \mathcal{S}_d$ where $\gamma = \lceil \frac{\log n}{\log\log n} \rceil + O(\frac{\log n}{(\log\log n)^{3/2}})$, $d = e^{\mathcal{O}(\sqrt{\log\log n})} \log n$ and $x = 5d + 1$. We lift the dimension such that $\mathbf{a}_i = (\mathbf{a}_i[0], \cdots, \mathbf{a}_i[\gamma+3])$ where $\mathbf{a}_i[\gamma+1] = \mathbf{a}_i[\gamma+2] = \mathbf{a}_i[\gamma+3] = 0$.
- Apply Lemma 3.3 again by setting $N = 4\gamma + 4$, we get another injection σ' such that $\sigma'(y) = o(\log^2 n) < x^2$ for $y \le 4\gamma + 4$.
- We have constructed in the proof of Lemma 3.4: $\mathbf{b}_i^0 = \mathbf{a}_i, \mathbf{b}_i^1, \mathbf{b}_i^2, \cdots, \mathbf{b}_i^{2\gamma+2} = \hat{\mathbf{b}}_k^{2\gamma+2}, \hat{\mathbf{b}}_k^{2\gamma+1}, \cdots, \hat{\mathbf{b}}_k^1, \hat{\mathbf{b}}_k^0 = \mathbf{a}_k$ where $k = \tau(i)$.
- Again, each vector $\mathbf{c}$ represents the polynomial $\sum_i \mathbf{c}[i]x^i$. Polynomials and vectors are used interchangeably.
- Let $\sigma_{max} = x^{\gamma+6} = n^{1+O(\frac{1}{\sqrt{\log\log n}})}$, and thus $\sigma_{max} > x \cdot \mathbf{b}_i^h \mathbf{x}$ and $\sigma_{max} > x \cdot \hat{\mathbf{b}}_i^h \mathbf{x}$ for any i, h, and also $\sigma_{max} > \sigma'(y)x^{\gamma+3}$ for any $y \le 4\gamma + 4$.

Construction of the scheduling instance. We shall construct two major classes of jobs, gap jobs and main jobs. Main jobs are divided into 5 types: dummy jobs, clause jobs, truth-assignment jobs, link jobs and variable jobs. The three types – truth-assignment, link and variable jobs – are further divided into sub-types, e.g., variable jobs are further divided into 4 sub-types (see Table 1). A gap job is defined as a fixed huge value $10^{14}\sigma_{max}$ subtracting several main jobs.

The processing time of each job can be expressed as a summation over three components: Type, Index and True/False. The type component of a main job is always of the form $10^j\sigma_{max}$ where $2 \le j \le 13$. Table 1 summarizes the value j for each kind of main job, e.g., the type-component of a variable job whose sub-type belongs to $\mathrm{V}_{\cdot,+,1}$ is $10^5\sigma_{max}$. The index-component of clause jobs, truth-assignment jobs and variable jobs is of the form $10\sigma(i)$ for some index i. Dummy jobs do not have index-component; Link jobs have much more complicated index-components, which will be specified in the following part of this subsection. Each main job has a true version and a false version. A gap job does not have a true/false version but only one unified version.

Define a function ζ that maps the (sub)-type of a main job to the exponent of 10 as indicated by Table 1, e.g., $\zeta(\mathrm{TR}_{\cdot,a}) = 11$. Now we provide the exact processing time of every job. In the following $\rho \in \{T, F\}$, $\iota \in \{+, -\}$.

\	Dummy	Clause	Truth-assignment				Link		Variable			
\	DM	CL.	$\mathrm{TR}_{\cdot,a}$	$\mathrm{TR}_{\cdot,b}$	$\mathrm{TR}_{\cdot,c}$	$\mathrm{TR}_{\cdot,d}$	$\mathrm{LN}_{\cdot,+}$	$\mathrm{LN}_{\cdot,-}$	$\mathrm{V}_{\cdot,+,1}$	$\mathrm{V}_{\cdot,+,2}$	$\mathrm{V}_{\cdot,-,1}$	$\mathrm{V}_{\cdot,-,2}$
$\zeta(\cdot)$	13	12	11	10	9	8	7	6	5	4	3	2

Table 1: Type-component of main jobs

• Variable jobs: 4 jobs $V^{\rho}_{i,+,1}$ and $V^{\rho}_{i,+,2}$ are constructed for the positive literal z_i, and 4 jobs $V^{\rho}_{i,-,1}$ and $V^{\rho}_{i,-,2}$ are for the negative literal $\neg z_i$.

$$s(V^{T}_{i,\iota,\kappa}) = 10^{\zeta(\mathrm{V}_{\cdot,\iota,\kappa})}\sigma_{max} + 10\sigma(i) + 1,$$
$$s(V^{F}_{i,\iota,\kappa}) = 10^{\zeta(\mathrm{V}_{\cdot,\iota,\kappa})}\sigma_{max} + 10\sigma(i) + 2, \quad \kappa = 1,2, \iota = +,-$$

• Truth-assignment jobs: 8 jobs $\mathrm{TR}^{\rho}_{i,a}$, $\mathrm{TR}^{\rho}_{i,b}$, $\mathrm{TR}^{\rho}_{i,c}$ and $\mathrm{TR}^{\rho}_{i,d}$ are constructed for every i.

$$s(\mathrm{TR}^{T}_{i,\kappa}) = 10^{\zeta(\mathrm{TR}_{\cdot,\kappa})}\sigma_{max} + 10\sigma(i) + 1.5,$$
$$s(\mathrm{TR}^{F}_{i,\kappa}) = 10^{\zeta(\mathrm{TR}_{\cdot,\kappa})}\sigma_{max} + 10\sigma(i) + 1. \quad \kappa = a,b,c,d$$

• Clause jobs: there are 3 clause jobs for every clause $cl_\ell \in C_1$ where $\ell \in \{2,5,\cdots,n-1\}$, with one CL^{T}_{ℓ} and two copies of CL^{F}_{ℓ}:

$$s(\mathrm{CL}^{T}_{\ell}) = 10^{\zeta(\mathrm{CL}_{\cdot})}\sigma_{max} + 10\sigma(\ell) + 2, \quad s(\mathrm{CL}^{F}_{\ell}) = 10^{\zeta(\mathrm{CL}_{\cdot})}\sigma_{max} + 10\sigma(\ell) + 1.$$

• Dummy jobs: there are $n + n/3$ true dummy jobs DM^{T} of processing time $10^{\zeta(\mathrm{DM})}\sigma_{max} + 1$, and $n - n/3$ false dummy jobs DM^{F} of processing time $10^{\zeta(\mathrm{DM})}\sigma_{max} + 2$.

• Link jobs: We create $4\gamma+4$ links jobs for each clause in C_2. Recall the vectors $\mathbf{b}^h_i$ and $\hat{\mathbf{b}}^h_i$ for $1 \le h \le 2\gamma+2$. For every clause $(z_i \oplus z_k) \in C_2$ and every $1 \le h \le 2\gamma+2$, we create two pairs of link jobs, $\mathrm{LN}^{T}_{i,h,+}$ and $\mathrm{LN}^{F}_{i,h,+}$, and $\mathrm{LN}^{T}_{k,h,-}$ and $\mathrm{LN}^{F}_{k,h,-}$ such that

$$s(\mathrm{LN}^{T}_{i,h,+}) = 10^{\zeta(\mathrm{LN}_{\cdot,+})}\sigma_{max} + 10\mathbf{b}^h(i)\mathbf{x} + 1, \quad s(\mathrm{LN}^{F}_{i,h,+}) = 10^{\zeta(\mathrm{LN}_{\cdot,+})}\sigma_{max} + 10\mathbf{b}^h_i\mathbf{x} + 2,$$

$$s(\mathrm{LN}^{T}_{k,h,-}) = 10^{\zeta(\mathrm{LN}_{\cdot,-})}\sigma_{max} + 10\hat{\mathbf{b}}^h(k)\mathbf{x} + 1, \quad s(\mathrm{LN}^{F}_{k,h,-}) = 10^{\zeta(\mathrm{LN}_{\cdot,-})}\sigma_{max} + 10\hat{\mathbf{b}}^h(i)\mathbf{x} + 2.$$

Let TR_A, TR_B, TR_C, TR_D be the set of jobs $\mathrm{TR}^{\rho}_{i,a}$, $\mathrm{TR}^{\rho}_{i,b}$, $\mathrm{TR}^{\rho}_{i,c}$ and $\mathrm{TR}^{\rho}_{i,d}$ respectively. Sometimes we may drop the superscript for simplicity, e.g., we use $\mathrm{TR}_{i,a}$ to represent $\mathrm{TR}^{T}_{i,a}$ or $\mathrm{TR}^{F}_{i,a}$. We construct gap jobs. There are 5 kinds of gap jobs.

• There are two gap jobs (variable-link jobs) $\theta_{\mathrm{LN},i,+}$ and $\theta_{\mathrm{V\text{-}L},i,-}$ for each variable z_i:

$$\begin{aligned} s(\theta_{\mathrm{V\text{-}L},i,+}) &= (10^{14} - 10^{7} - 10^{4})\sigma_{max} - 10\left(\mathbf{a}_i[0] + 2\sum_{j=1}^{\gamma}\mathbf{a}_i[j]x^j + (F_0(\mathbf{a}_i))[0]\cdot x^{\gamma+1} + \sigma'(1)x^{\gamma+2}\right) - 3 \\ &= \left(10^{14} - 10^{\zeta(\mathrm{LN}_{\cdot,+})} - 10^{\zeta(V_{\cdot,+,2})}\right)\sigma_{max} - 10(\mathbf{a}_i + \mathbf{b}^1_i)\mathbf{x} - 3 \\ s(\theta_{\mathrm{V\text{-}L},i,-}) &= (10^{14} - 10^{6} - 10^{2})\sigma_{max} - 10\left(\mathbf{a}_i[0] + 2\sum_{j=1}^{\gamma}\mathbf{a}_i[j]x^j + (\hat{F}(\mathbf{a}_i))[0]\cdot x^{\gamma+1} + \sigma'(1)x^{\gamma+2}\right) - 3 \\ &= \left(10^{14} - 10^{\zeta(\mathrm{LN}_{\cdot,-})} - 10^{\zeta(V_{\cdot,-,2})}\right)\sigma_{max} - 10\left(\mathbf{a}_i + \hat{\mathbf{b}}^1_i\right)\mathbf{x} - 3 \end{aligned}$$

• There are $4\gamma + 3$ gap jobs (link-link jobs), $\theta_{\mathrm{L\text{-}L},i,h,+}$ and $\theta_{\mathrm{L\text{-}L},i,h,-}$ and $\theta_{\mathrm{L\text{-}L},i,+,-}$ for every $1 \le i \le n$. For $h = 1,2,\cdots,2\gamma+1$, we define

$$\begin{aligned} s(\theta_{\mathrm{L\text{-}L},i,h,+}) &= \left(10^{14} - 2\times 10^{\zeta(\mathrm{LN}_{\cdot,+})}\right)\sigma_{max} - 10\left(\mathbf{b}^h_i\mathbf{x} + \mathbf{b}^{h+1}_i\right)\mathbf{x} - 3 \\ s(\theta_{\mathrm{L\text{-}L},i,h,-}) &= \left(10^{14} - 2\times 10^{\zeta(\mathrm{LN}_{\cdot,-})}\right)\sigma_{max} - 10\left(\hat{\mathbf{b}}^h_i\mathbf{x} + \hat{\mathbf{b}}^{h+1}_i\right)\mathbf{x} - 3 \end{aligned}$$

Additionally, we define

$$s(\theta_{\text{L-L},i,+,-}) = \left(10^{14} - 10^{\zeta(\text{LN}_{\cdot,+})} - 10^{\zeta(\text{LN}_{\cdot,-})}\right)\sigma_{max} - 2 \times 10\mathbf{b}_i^{2\gamma+2}\mathbf{x} - 3$$

Here recall that $\mathbf{b}_i^{2\gamma+2} = \hat{\mathbf{b}}_{\tau(i)}^{2\gamma+2}$.

• There are three gap jobs (variable-clause-dummy jobs) for each $cl_\ell \in C_1$ ($\ell \in \{2, 5, \cdots, n-1\}$): for $i = \ell - 1, \ell, \ell + 1$, if $z_i \in cl_\ell$, we construct $\theta_{\text{V-C-D},\ell,i,+}$, otherwise $\neg z_i \in cl_\ell$, and we construct $\theta_{\text{V-C-D},\ell,i,-}$:

$$s(\theta_{\text{V-C-D},\ell,i,+}) = \left(10^{14} - 10^{\zeta(\text{DM})} - 10^{\zeta(\text{CL}_\cdot)} - 10^{\zeta(V_{\cdot,+,1})}\right)\sigma_{max} - 10(\sigma(\ell) + \sigma(i)) - 4,$$
$$s(\theta_{\text{V-C-D},\ell,i,-}) = \left(10^{14} - 10^{\zeta(\text{DM})} - 10^{\zeta(\text{CL}_\cdot)} - 10^{\zeta(V_{\cdot,-,1})}\right)\sigma_{max} - 10(\sigma(\ell) + \sigma(i)) - 4.$$

• There is one gap job (variable-dummy job) for each variable. Notice that each variable appears exactly once in clauses of C_1, if z_i appears in C_1, we construct $\theta_{\text{V-D},i,-}$. Otherwise, we construct $\theta_{\text{V-D},i,+}$ instead.

$$s(\theta_{\text{V-D},i,+}) = \left(10^{14} - 10^{\zeta(\text{DM})} - 10^{\zeta(V_{\cdot,+,1})}\right)\sigma_{max} - 10\sigma(i) - 3,$$
$$s(\theta_{\text{V-D},i,-}) = \left(10^{14} - 10^{\zeta(\text{DM})} - 10^{\zeta(V_{\cdot,-,1})}\right)\sigma_{max} - 10\sigma(i) - 3.$$

Thus, for each clause cl_ℓ and $i = \ell-1, \ell, \ell+1$, either $\theta_{\text{V-D},i,+}$ and $\theta_{\text{V-C-D},\ell,i,-}$ exist, or $\theta_{\text{V-D},i,-}$ and $\theta_{\text{V-C-D},\ell,i,+}$ exist.

• There are four gap jobs (variable-truth jobs) for each variable z_i, namely $\theta_{\text{V-T},i,a,c}$, $\theta_{\text{V-T},i,b,d}$, $\theta_{\text{V-T},i,a,d}$ and $\theta_{\text{V-T},i,b,c}$:

$$s(\theta_{\text{V-T},i,a,c}) = \left(10^{14} - 10^{\zeta(V_{\cdot,+,1})} - 10^{\zeta(\text{TR}_{\cdot,a})} - 10^{\zeta(\text{TR}_{\cdot,c})}\right)\sigma_{max} - 30\sigma(i) - 4,$$
$$s(\theta_{\text{V-T},i,b,d}) = \left(10^{14} - 10^{\zeta(V_{\cdot,+,2})} - 10^{\zeta(\text{TR}_{\cdot,b})} - 10^{\zeta(\text{TR}_{\cdot,d})}\right)\sigma_{max} - 30\sigma(i) - 4,$$
$$s(\theta_{\text{V-T},i,a,d}) = \left(10^{14} - 10^{\zeta(V_{\cdot,-,1})} - 10^{\zeta(\text{TR}_{\cdot,a})} - 10^{\zeta(\text{TR}_{\cdot,d})}\right)\sigma_{max} - 30\sigma(i) - 4,$$
$$s(\theta_{\text{V-T},i,b,c}) = \left(10^{14} - 10^{\zeta(V_{\cdot,-,2})} - 10^{\zeta(\text{TR}_{\cdot,b})} - 10^{\zeta(\text{TR}_{\cdot,c})}\right)\sigma_{max} - 30\sigma(i) - 4,$$

Overall, we have constructed $2\gamma n + 8n$ gap jobs. We also construct $2\gamma n + 8n$ machines. The following Table 2 summarizes the processing times of all jobs.

G Proof of Theorem 3.1

The proof is carried out in 4 steps. We first show in Section G.1 that every job in the constructed instance has a unique processing time. This allows us to refer to a job by its symbol (e.g., $V_{i,+,1}^T$) as well as by its processing time. Next, we show in Section G.2 that if a significant fraction of clauses in the 3SAT$'$ instance are satisfiable, then the constructed scheduling instance admits a solution with a small objective value. Next, we show in Section G.3 that if any truth-assignment for the 3SAT$'$ instance will leave a significant fraction of clauses unsatisfied, then the constructed scheduling instance does not admit a solution with a small objective value. Finally, we are able to prove the correctness of our reduction in Section G.4 by leveraging the above two facts.

G.1 Uniqueness of job processing times We claim that the processing time of each job we create is unique, whereas there is a one-to-one correspondence between the symbol of a job and its processing time. To see the claim, consider Table 2. It suffices to compare the processing time of jobs within each subtype. Given that σ is an injection, it is easy to see that the processing time of each variable job, truth-assignment job, clause job, variable-dummy job and variable-truth job is unique. For variable-clause-dummy jobs, by property 4 of Lemma 3.3 we know the sum $\sigma(\ell)+\sigma(i)$ for $i = \ell-1, \ell, \ell+1$ is unique. The uniqueness of link jobs follows from the uniqueness of $\mathbf{b}_i^h$'s from Lemma 3.6. The uniqueness of link-link jobs follows from the uniqueness of the summation $\mathbf{b}_i^h + \mathbf{b}_i^{h+1}$ from Lemma 3.7.

Job-type	Sub-type	Type-component	Index-component	T/F (T)	T/F (F)
Variable	$V_{i,+,1}$	$10^{\zeta(\mathrm{V}_{\cdot,+,1})}\sigma_{max}$	$10\sigma(i)$	1	2
	$V_{i,+,2}$	$10^{\zeta(\mathrm{V}_{\cdot,+,2})}\sigma_{max}$	$10\sigma(i)$	1	2
	$V_{i,-,1}$	$10^{\zeta(\mathrm{V}_{\cdot,-,1})}\sigma_{max}$	$10\sigma(i)$	1	2
	$V_{i,-,2}$	$10^{\zeta(\mathrm{V}_{\cdot,-,2})}\sigma_{max}$	$10\sigma(i)$	1	2
Truth-assignment	$\mathrm{TR}_{i,a}$	$10^{\zeta(\mathrm{TR}_{\cdot,a})}\sigma_{max}$	$10\sigma(i)$	1.5	1
	$\mathrm{TR}_{i,b}$	$10^{\zeta(\mathrm{TR}_{\cdot,b})}\sigma_{max}$	$10\sigma(i)$	1.5	1
	$\mathrm{TR}_{i,c}$	$10^{\zeta(\mathrm{TR}_{\cdot,c})}\sigma_{max}$	$10\sigma(i)$	1.5	1
	$\mathrm{TR}_{i,d}$	$10^{\zeta(\mathrm{TR}_{\cdot,d})}\sigma_{max}$	$10\sigma(i)$	1.5	1
Clause	CL_{ℓ}	$10^{\zeta(\mathrm{CL}_{\cdot})}\sigma_{max}$	$10\sigma(\ell)$	2	1
Dummy	DM	$10^{\zeta(\mathrm{DM})}\sigma_{max}$	0	1	2
Link $h \in \{0,1,\cdots,2\gamma+2\}$	$\mathrm{LN}_{i,h,+}$	$10^{\zeta(\mathrm{LN}_{\cdot,+})}\sigma_{max}$	$10\mathbf{b}_i^h\mathbf{x}$	1	2
	$\mathrm{LN}_{i,h,-}$	$10^{\zeta(\mathrm{LN}_{\cdot,-})}\sigma_{max}$	$10\mathbf{b}_i^h\mathbf{x}$	1	2
Variable-Link	$\theta_{\text{V-L},i,+}$	$(10^{14}-10^{\zeta(\mathrm{LN}_{\cdot,+})}-10^{\zeta(V_{\cdot,+,2})})\sigma_{max}$	$-10(\mathbf{a}_i+\mathbf{b}_i^1)\mathbf{x}$	-3	
	$\theta_{\text{V-L},i,-}$	$(10^{14}-10^{\zeta(\mathrm{LN}_{\cdot,-})}-10^{\zeta(V_{\cdot,-,2})})\sigma_{max}$	$-10(\mathbf{a}_i+\hat{\mathbf{b}}_i^1)\mathbf{x}$	-3	
Link-Link $h \in \{1,\cdots,2\gamma+1\}$	$\theta_{\text{L-L},i,h,+}$	$(10^{14}-2\times 10^{\zeta(\mathrm{LN}_{\cdot,+})})\sigma_{max}$	$-10(\mathbf{b}_i^h+\mathbf{b}_i^{h+1})\mathbf{x}$	-3	
	$\theta_{\text{L-L},i,h,-}$	$(10^{14}-2\times 10^{\zeta(\mathrm{LN}_{\cdot,-})})\sigma_{max}$	$-10(\mathbf{b}_i^h+\hat{\mathbf{b}}_i^{h+1})\mathbf{x}$	-3	
	$\theta_{\text{L-L},i,+,-}$	$(10^{14}-10^{\zeta(\mathrm{LN}_{\cdot,+})}-10^{\zeta(\mathrm{LN}_{\cdot,-})})\sigma_{max}$	$-20\mathbf{b}_i^{2\gamma+2}\mathbf{x}$	-3	
Variable-Clause -Dummy, $\lvert i-\ell\rvert \le 1$	$\theta_{\text{V-C-D},\ell,i,+}$	$(10^{14}-10^{\zeta(\mathrm{DM})}-10^{\zeta(\mathrm{CL}_{\cdot})}-10^{\zeta(V_{\cdot,+,1})})\sigma_{max}$	$-10(\sigma(\ell)+\sigma(i))$	-4	
	$\theta_{\text{V-C-D},\ell,i,-}$	$(10^{14}-10^{\zeta(\mathrm{DM})}-10^{\zeta(\mathrm{CL}_{\cdot})}-10^{\zeta(V_{\cdot,-,1})})\sigma_{max}$	$-10(\sigma(\ell)+\sigma(i))$	-4	
Variable-Dummy	$\theta_{\text{V-D},i,+}$	$(10^{14}-10^{\zeta(\mathrm{DM})}-10^{\zeta(V_{\cdot,+,1})})\sigma_{max}$	$-10\sigma(i)$	-3	
	$\theta_{\text{V-D},i,-}$	$(10^{14}-10^{\zeta(\mathrm{DM})}-10^{\zeta(V_{\cdot,-,1})})\sigma_{max}$	$-10\sigma(i)$	-3	
Variable-Truth	$\theta_{\text{V-T},i,a,c}$	$(10^{14}-10^{\zeta(V_{\cdot,+,1})}-10^{\zeta(\mathrm{TR}_{\cdot,a})}-10^{\zeta(\mathrm{TR}_{\cdot,c})})\sigma_{max}$	$-30\sigma(i)$	-4	
	$\theta_{\text{V-T},i,b,d}$	$(10^{14}-10^{\zeta(V_{\cdot,+,2})}-10^{\zeta(\mathrm{TR}_{\cdot,b})}-10^{\zeta(\mathrm{TR}_{\cdot,d})})\sigma_{max}$	$-30\sigma(i)$	-4	
	$\theta_{\text{V-T},i,a,d}$	$(10^{14}-10^{\zeta(V_{\cdot,-,1})}-10^{\zeta(\mathrm{TR}_{\cdot,a})}-10^{\zeta(\mathrm{TR}_{\cdot,d})})\sigma_{max}$	$-30\sigma(i)$	-4	
	$\theta_{\text{V-T},i,b,c}$	$(10^{14}-10^{\zeta(V_{\cdot,-,2})}-10^{\zeta(\mathrm{TR}_{\cdot,b})}-10^{\zeta(\mathrm{TR}_{\cdot,c})})\sigma_{max}$	$-30\sigma(i)$	-4	

Table 2: Job processing times

G.2 3SAT′ to Scheduling The goal of this subsection is to prove the following lemma.

LEMMA G.1. *If there are at most ϑn clauses which are not satisfied, then the constructed scheduling instance admits a feasible schedule with objective value at most* $(10^{14}\sigma_{max})^q(2\gamma n+8n)+\vartheta n\cdot\frac{q(q-1)}{2}(10^{14}\sigma_{max})^{q-2}+o(n\sigma_{max}^{q-2})$.

Recall that every main job, except the clause job, admits a true copy and false copy, while the clause job admits a true copy and two false copies. We first ignore the true/false version of jobs and schedule them according to Table 3, where each row represents jobs that are scheduled on one machine.

We show that if we schedule according to Table 3, then every job has been scheduled (ignoring the superscripts T or F, which will be determined later). It is obvious that every gap job is scheduled. For simplicity, we abuse the notation a bit by using the symbol of a gap job to denote the machine on which it is scheduled.

• Consider clause jobs. Recall that for each clause cl_ℓ and $i=\ell-1,\ell,\ell+1$, we either construct $\theta_{\text{V-D},i,-}$ and $\theta_{\text{V-C-D},\ell,i,+}$ if the positive literal z_i occurs in C_1, or construct $\theta_{\text{V-D},i,+}$ and $\theta_{\text{V-C-D},\ell,i,-}$ if the negative literal $\neg z_i$ occurs in C_1. Hence the three copies of job CL_ℓ appear on machine $\theta_{\text{V-C-D},\ell,\ell-1,+}$ or $\theta_{\text{V-C-D},\ell,\ell-1,-}$, machine $\theta_{\text{V-C-D},\ell,\ell,+}$ or $\theta_{\text{V-C-D},\ell,\ell,-}$, and machine $\theta_{\text{V-C-D},\ell,\ell+1,+}$ or $\theta_{\text{V-C-D},\ell,\ell+1,-}$. Thus, all three copies of a clause job are scheduled.

• Consider truth-assignment jobs. There are two copies of $\mathrm{TR}_{i,a}$, $\mathrm{TR}_{i,b}$, $\mathrm{TR}_{i,c}$ and $\mathrm{TR}_{i,d}$. It is easy to see that all of them are scheduled on machines $\theta_{\text{V-T},i,a,c}$, $\theta_{\text{V-T},i,b,d}$, $\theta_{\text{V-T},i,a,d}$ and $\theta_{\text{V-T},i,b,c}$.

• Consider variable jobs. There are two copies of $V_{i,+,1}$, $V_{i,+,2}$, $V_{i,-,1}$ and $V_{i,-,2}$. It is easy to see that one copy of them are scheduled on machines $\theta_{\text{V-T},i,a,c}$, $\theta_{\text{V-T},i,b,d}$, $\theta_{\text{V-T},i,a,d}$ and $\theta_{\text{V-T},i,b,c}$. One copy of $V_{i,+,2}$ and $V_{i,-,2}$ are scheduled on machines $\theta_{\text{V-L},i,+}$ and $\theta_{\text{V-L},i,-}$. If machines $\theta_{\text{V-D},i,-}$ and $\theta_{\text{V-C-D},\ell,i,+}$ exist (when the positive

Variable-Link	$\theta_{\text{V-L},i,+}$	$V_{i,+,2}$	$\text{LN}_{i,1,+}$	\
	$\theta_{\text{V-L},i,-}$	$V_{i,-,2}$	$\text{LN}_{i,1,-}$	\
Link-Link $h \in \{1,2,\cdots,2\gamma+1\}$	$\theta_{\text{L-L},i,h,+}$	$\text{LN}_{i,h,+}$	$\text{LN}_{i,h+1,+}$	\
	$\theta_{\text{L-L},i,h,-}$	$\text{LN}_{i,h,-}$	$\text{LN}_{i,h+1,-}$	\
	$\theta_{\text{L-L},i,+,-}$	$\text{LN}_{i,2\gamma+2,+}$	$\text{LN}_{\tau(i),2\gamma+2,-}$	\
Variable-Clause-Dummy $\lvert i-\ell\rvert \le 1$	$\theta_{\text{V-C-D},\ell,i,+}$	$V_{i,+,1}$	CL_ℓ	DM
	$\theta_{\text{V-C-D},\ell,i,-}$	$V_{i,-,1}$	CL_ℓ	DM
Variable-Dummy	$\theta_{\text{V-D},i,+}$	$V_{i,+,1}$	DM	\
	$\theta_{\text{V-D},i,-}$	$V_{i,-,1}$	DM	\
Variable-Truth	$\theta_{\text{V-T},i,a,c}$	$V_{i,+,1}$	$\text{TR}_{i,a}$	$\text{TR}_{i,c}$
	$\theta_{\text{V-T},i,b,d}$	$V_{i,+,2}$	$\text{TR}_{i,b}$	$\text{TR}_{i,d}$
	$\theta_{\text{V-T},i,a,d}$	$V_{i,-,1}$	$\text{TR}_{i,a}$	$\text{TR}_{i,d}$
	$\theta_{\text{V-T},i,b,c}$	$V_{i,-,2}$	$\text{TR}_{i,b}$	$\text{TR}_{i,c}$

Table 3: SAT to Scheduling – Jobs scheduled on each machine

literal z_i occurs in C_1), then $V_{i,-,1}$ and $V_{i,+,1}$ are scheduled on them respectively; otherwise machines $\theta_{\text{V-D},i,+}$ and $\theta_{\text{V-C-D},\ell,i,-}$ exist (the negative literal $\neg z_i$ occurs in C_1), then $V_{i,+,1}$ and $V_{i,-,1}$ are scheduled on them respectively.

• Consider link jobs. There are two copies of $\text{LN}_{i,h,+}$ (or $\text{LN}_{i,h,-}$) for $1 \le h \le 2\gamma+2$. Let $\iota \in \{+,-\}$. The two copies of $\text{LN}_{i,1,\iota}$ are scheduled on machines $\theta_{\text{V-L},i,\iota}$ and $\theta_{\text{L-L},i,1,\iota}$. The two copies of $\text{LN}_{i,h,\iota}$ are scheduled on $\theta_{\text{L-L},i,h,\iota}$ and $\theta_{\text{L-L},i,h+1,\iota}$ for $2 \le h \le 2\gamma+1$. The two copies of $\text{LN}_{i,2\gamma+2,+}$ are scheduled on machines $\theta_{\text{L-L},2\gamma+1,+}$ and $\theta_{\text{L-L},i,+,-}$, and the two copies of $\text{LN}_{i,2\gamma+2,-}$ are scheduled on machines $\theta_{\text{L-L},2\gamma+1,-}$ and $\theta_{\text{L-L},\tau^{-1}(i),+,-}$, where τ^{-1} is the inverse of the mapping τ (note that τ^{-1} exists since τ is one-to-one).

• Consider dummy jobs. There are in total $2n$ dummy jobs. It is obvious that for every i, 2 dummy jobs are scheduled on machines $\theta_{\text{V-C-D},\ell,i,+}$, $\theta_{\text{V-D},i,-}$ or machines $\theta_{\text{V-C-D},\ell,i,-}$, $\theta_{\text{V-D},i,+}$.

Next, we consider the load of every machine. According to Table 2, it is easy to verify that if we sum up the type-component of jobs on each machine, it becomes $10^{14}\sigma_{max}$; if we sum up the index-component of jobs on each machine, it becomes 0. Now we consider the T/F-component of jobs. It is easy to verify that the T/F-components of all jobs add up to 0, hence we have the following direct observation.

OBSERVATION 5. *The total processing time of all jobs add up to* $10^{14}\sigma_{max} \cdot (2\gamma n + 8n)$.

Variable-Link	$\theta_{\text{V-L},i,+}$	$V^T_{i,+,2}$	$\text{LN}^F_{i,1,+}$	\
	$\theta_{\text{V-L},i,-}$	$V^F_{i,-,2}$	$\text{LN}^T_{i,1,-}$	\
Link-Link $h \in \{1,2,\cdots,2\gamma+1\}$	$\theta_{\text{L-L},i,h,+}$	$\text{LN}^F_{i,h,+}$	$\text{LN}^T_{i,h+1,+}$	\
	$\theta_{\text{L-L},i,h,-}$	$\text{LN}^F_{i,h,-}$	$\text{LN}^T_{i,h+1,-}$	\
	$\theta_{\text{L-L},i,+,-}$	$\text{LN}^T_{i,2\gamma+2,+}$	$\text{LN}^*_{\tau(i),2\gamma+2,-}$	\
Variable-Clause-Dummy & Variable-Dummy Case 1: positive literal $z_i \in C_1$	$\theta_{\text{V-C-D},\ell,i,+}$	$V^T_{i,+,1}$	CL^*_ℓ	DM*
	$\theta_{\text{V-D},i,-}$	$V^F_{i,-,1}$	DM*	\
Variable-Clause-Dummy & Variable-Dummy Case 2: negative literal $\neg z_i \in C_1$	$\theta_{\text{V-C-D},i,-}$	$V^F_{i,-,1}$	CL^*_ℓ	DM*
	$\theta_{\text{V-D},i,+}$	$V^T_{i,+,1}$	DM*	\
Variable-Truth	$\theta_{\text{V-T},i,a,c}$	$V^F_{i,+,1}$	$\text{TR}^F_{i,a}$	$\text{TR}^F_{i,c}$
	$\theta_{\text{V-T},i,b,d}$	$V^F_{i,+,2}$	$\text{TR}^F_{i,b}$	$\text{TR}^F_{i,d}$
	$\theta_{\text{V-T},i,a,d}$	$V^T_{i,-,1}$	$\text{TR}^T_{i,a}$	$\text{TR}^T_{i,d}$
	$\theta_{\text{V-T},i,b,c}$	$V^T_{i,-,2}$	$\text{TR}^T_{i,b}$	$\text{TR}^T_{i,c}$

Table 4: Scheduling of Truth/False types of jobs if the variable z_i is true

Variable-Link	$\theta_{\text{V-L},i,+}$	$V^F_{i,+,2}$	$\text{LN}^T_{i,1,+}$	\
	$\theta_{\text{V-L},i,-}$	$V^T_{i,-,2}$	$\text{LN}^F_{i,1,-}$	\
Link-Link $h \in \{1,2,\cdots,2\gamma+1\}$	$\theta_{\text{L-L},i,h,+}$	$\text{LN}^T_{i,h,+}$	$\text{LN}^F_{i,h+1,+}$	\
	$\theta_{\text{L-L},i,h,-}$	$\text{LN}^T_{i,h,-}$	$\text{LN}^F_{i,h+1,-}$	\
	$\theta_{\text{L-L},i,+,-}$	$\text{LN}^F_{i,2\gamma+2,+}$	$\text{LN}^*_{\tau(i),2\gamma+2,-}$	\
Variable-Clause-Dummy & Variable-Dummy Case 1: positive literal $z_i \in C_1$	$\theta_{\text{V-C-D},\ell,i,+}$	$V^F_{i,+,1}$	CL^*_ℓ	DM^*
	$\theta_{\text{V-D},i,-}$	$V^T_{i,-,1}$	DM^*	\
Variable-Clause-Dummy & Variable-Dummy Case 2: negative literal $\neg z_i \in C_1$	$\theta_{\text{V-C-D},i,-}$	$V^T_{i,-,1}$	CL^*_ℓ	DM^*
	$\theta_{\text{V-D},i,+}$	$V^F_{i,+,1}$	DM^*	\
Variable-Truth	$\theta_{\text{V-T},i,a,c}$	$V^T_{i,+,1}$	$\text{TR}^T_{i,a}$	$\text{TR}^T_{i,c}$
	$\theta_{\text{V-T},i,b,d}$	$V^T_{i,+,2}$	$\text{TR}^T_{i,b}$	$\text{TR}^T_{i,d}$
	$\theta_{\text{V-T},i,a,d}$	$V^F_{i,-,1}$	$\text{TR}^F_{i,a}$	$\text{TR}^F_{i,d}$
	$\theta_{\text{V-T},i,b,c}$	$V^F_{i,-,2}$	$\text{TR}^F_{i,b}$	$\text{TR}^F_{i,c}$

Table 5: Scheduling of Truth/False types of jobs if the variable z_i is false

Consider the truth-assignment of I_{sat}. If the variable z_i is true, then we determine the true/false version of main jobs according to Table 4. Otherwise the variable z_i is false in the assignment, then we flip the True/False version of all jobs in Table 4, i.e., we schedule according to Table 5. It is easy to see that in each row of Table 4, if there is no job with a superscript of $*$, then their T/F-components sum up to 0, i.e., the load of this machine is exactly $10^{14}\sigma_{max}$. We call the current schedule a semi-schedule. It remains to determine the true/false version of jobs with the superscript $*$.

• Consider link-link machines. We only need to consider machines $\theta_{\text{L-L},i,+,-}$. The T/F-type of the job $\text{LN}_{i,2\gamma+2,+}$ has already been decided based on the true/false of variable z_i. Consider the other job $\text{LN}_{\tau(i),2\gamma+2,-}$ scheduled on this machine. Notice that based on the true/false of the variable $z_{\tau(i)}$, one copy of $\text{LN}_{\tau(i),2\gamma+2,-}$ is scheduled on $\theta_{\text{L-L},\tau(i),2\gamma+1,-}$, and the remaining copy is scheduled on $\theta_{\text{L-L},i,+,-}$. If $z_{\tau(i)}$ is true, the remaining copy is $\text{LN}^F_{\tau(i),2\gamma+2,-}$; otherwise, the remaining copy is $\text{LN}^T_{\tau(i),2\gamma+2,-}$. Hence, we have the following observation:

– if variables z_i is true and $z_{\tau(i)}$ is false, then $\text{LN}^T_{i,2\gamma+2,+}$ and $\text{LN}^T_{\tau(i),2\gamma+2,-}$ are on this machine, whereas the load is $10^{14}\sigma_{max}-1$;

– if variables z_i is false and $z_{\tau(i)}$ is true, then $\text{LN}^F_{i,2\gamma+2,+}$ and $\text{LN}^F_{\tau(i),2\gamma+2,-}$ are on this machine, whereas the load is $10^{14}\sigma_{max}+1$;

– if variables z_i and $z_{\tau(i)}$ are both true or both false, then one of $\text{LN}_{i,2\gamma+2,+}$ and $\text{LN}_{\tau(i),2\gamma+2,-}$ is true and the other is false, whereas the load is $10^{14}\sigma_{max}$.

The above observation leads to the following claim.

Claim 2. *The load of machine $\theta_{L\text{-}L,i,+,-}$ is $10^{14}\sigma_{max}$ if the clause $(z_i \oplus \neg z_{\tau(i)})$ is satisfied, and is $10^{14}\sigma_{max} \pm 1$ otherwise.*

• Consider variable-clause-dummy and variable-dummy machines. Notice that there is one true copy and two false copies of CL_ℓ, scheduled on machines $\theta_{\text{V-C-D},\ell,i,\kappa_{i-\ell}}$ where $i \in \{\ell-1,\ell,\ell+1\}$ and $\kappa_{i-\ell} \in \{+,-\}$. If there exists at least one $i = i^*$ such that $V^T_{i,\kappa_{i^*-\ell},1}$ is on machine $\theta_{\text{V-C-D},\ell,i^*,\kappa_{i^*-\ell}}$, then we schedule CL^T_ℓ on machine $\theta_{\text{V-C-D},\ell,i^*,\kappa_{i^*-\ell}}$, and schedule the two copies of CL^F_ℓ on the remaining two machines, respectively. Otherwise, we schedule CL^T_ℓ on machine $\theta_{\text{V-C-D},\ell,\ell-1,\kappa_{-1}}$ and the two false copies CL^F_ℓ on machines $\theta_{\text{V-C-D},\ell,i,\kappa_{i-\ell}}$ where $i = \ell, \ell+1$.

Finally, we determine the true/false version of dummy jobs on variable-clause-dummy and variable-dummy machines. Recall that there are $n + n/3$ true dummy and $n - n/3$ false dummy jobs.

On variable-clause-dummy machines, if the clause job is true, schedule a true dummy job. Otherwise, the clause job is false, then if the variable job is true (or false), schedule a false (or true) dummy job.

On variable-dummy machines, we schedule dummy jobs in the following way. A false variable job is always scheduled with a true dummy job. For true variable jobs, we first partition the indices of variables, $\{1, 2, \cdots, n\}$, into two subsets S_1, S_2 such that

$$S_1 = \{i : \text{On machine } \theta_{\text{V-C-D},\ell,i,\kappa_{i-\ell}} \text{ there is a true clause job and a false variable job}\},$$

and S_2 consists of the remaining indices. On machine $\theta_{\text{V-D},i,+}$ or $\theta_{\text{V-D},i,-}$ where $i \in S_1$ and the variable job is true, we schedule a true dummy job; on machine $\theta_{\text{V-D},i,+}$ or $\theta_{\text{V-D},i,-}$ where $i \notin S_1$ and the variable job is true, we schedule a false dummy job.

Consider the true/false versions of all two jobs on a variable-dummy machine and use $(T/F, T/F)$ to denote the true/false version of the two jobs in the order of variable job, dummy job. Then the above scheduling can be restated as follows. A variable-dummy machine $\theta_{\text{V-D},i,+}$ or $\theta_{\text{V-D},i,+}$ is:

- (F, T), if a false variable job is on it;
- (T, F), if a true variable job is on it and $i \notin S_1$;
- (T, T), if a true variable job is on it and $i \in S_1$.

Hence there are in total three kinds of variable-dummy machines $(F, T), (T, T), (T, F)$.

Now we check the total number of true and false dummy jobs scheduled in the above way. Similarly we consider the true/false versions of all three jobs on a variable-clause-dummy machine and use $(T/F, T/F, T/F)$ to denote the true/false versions of the three jobs in the order of variable job, clause job and dummy job, then there are in total four kinds of variable-clause-dummy machines: $(F, T, T), (T, T, T), (T, F, F), (F, F, T)$. Let $\sharp(T/F, T/F, T/F)$ and $\sharp(T/F, T/F)$ be the number of machines of each kind. Then we have the following observations:

$$\sharp(F, T, T) = |S_1| \tag{G.5a}$$
$$\sharp(F, T, T) + \sharp(T, T, T) = n/3 \tag{G.5b}$$
$$\sharp(T, F, F) + \sharp(F, F, T) = 2n/3 \tag{G.5c}$$
$$\sharp(F, T) + \sharp(T, T) + \sharp(T, F) = n \tag{G.5d}$$
$$\sharp(F, T, T) + \sharp(F, F, T) + \sharp(F, T) = n \tag{G.5e}$$
$$\sharp(T, T) = |S_1| \tag{G.5f}$$

Here Eq (G.5a) follows from the definition of S_1. Eq (G.5b) follows from the fact that there are in total $n/3$ true clause jobs. Eq (G.5c) follows from the fact that there are in total n clause jobs, and hence $2n/3$ false clause jobs. Eq (G.5d) follows from the fact that there are in total n variable-dummy machines. Eq (G.5e) follows from the fact that there are in total n false variable jobs. We now explain Eq (G.5f). Notice that for each i there are in total 8 variable jobs (i.e., $V_{i,\cdot,\cdot}$), 4 true copies and 4 false copies. Among them 2 true and 2 false copies are scheduled on variable-truth machines, 1 true and 1 false copies are scheduled on variable-link machines (see Table 4). Hence, 1 true and 1 false copies are scheduled on variable-clause-dummy and variable-dummy machines. For any i, if the false (or true) variable job $V_{i,\cdot,\cdot}$ is scheduled on a variable-clause-dummy machine, then the remaining true (or false) variable job is scheduled on a variable-dummy machine. Now consider the set of all i's where the true variable job $V_{i,\cdot,\cdot}$ is scheduled with a true dummy job on a variable-dummy machine and let it be S_3. According to the way we schedule, on machine $\theta_{\text{V-D},i,+}$ or $\theta_{\text{V-D},i,-}$, we schedule a true variable job and a true dummy job only if $i \in S_1$ (otherwise, either the variable job or the dummy job is false), hence $S_3 \subseteq S_1$. Meanwhile, for any $i \in S_1$, we know the false variable job $V_{i,\cdot,\cdot}$ is scheduled on a variable-clause-dummy machine, whereas the true variable job must be scheduled on a variable-dummy machine, this implies that any $i \in S_1$ also satisfies that $i \in S_3$. Hence $S_1 = S_3$ and Eq (G.5f) is true.

The total number of true dummy jobs scheduled equals $\sharp(F, T, T) + \sharp(T, T, T) + \sharp(F, F, T) + \sharp(F, T) + \sharp(T, T) = n + \sharp(T, T, T) + \sharp(T, T) = n + n/3 - |S_1| + |S_1| = 4n/3$. Similarly, we can show the total number of false jobs scheduled equals $2n/3$. Hence, our way of scheduling dummy jobs is feasible.

Now we check the load of every variable-clause-dummy machines and variable-dummy machines. It is easy to verify that for a variable-clause-dummy machine, if its kind is (T, T, T), or (T, F, F), or (F, F, T), then its load is $10^{14}\sigma_{max}$; if its kind is (F, T, T), then its load is $10^{14}\sigma_{max} + 1$. For a variable-dummy machine, if its kind is (F, T) or (T, F), then its load is $10^{14}\sigma_{max}$; if its kind is (T, T), then its load is $10^{14}\sigma_{max} - 1$.

Notice that for every $1 \le i \le n$, the variable-dummy machine $\theta_{\text{V-D},i,\cdot}$ is of (T,T) if and only if the variable-clause-dummy machine $\theta_{\text{V-C-D},\ell,i,\cdot}$ is of (F,T,T). Recall that we always try to schedule the true clause job CL_ℓ^T with a true variable job, if possible. Hence, CL_ℓ^T is scheduled with a false variable job if and only if all the three variable jobs scheduled on variable-clause-dummy machines, i.e., $V_{\ell-1,\kappa_{-1},1}$, $V_{\ell,\kappa_0,1}$ and $V_{\ell+1,\kappa_1,1}$, are all false where $\kappa_{-1},\kappa_0,\kappa_1 \in \{+,-\}$. Consider $V_{\ell-1,\kappa_{-1},1}$. If $\kappa_{-1}=+$, then $\theta_{\text{V-C-D},\ell,\ell-1,+}$ exists, indicating case 1 of Table 4 or Table 5 occurs, i.e., the positive literal $z_{\ell-1}$ is in clause $cl_\ell \in C_1$. Furthermore, as $V^F_{\ell-1,+,1}$ is scheduled on the variable-clause-dummy machine, the scheduling follows Table 5, the variable $z_{\ell-1}$ is false in the assignment of I_{sat}. That is, cl_ℓ is not satisfied by $z_{\ell-1}$. Similarly, we can show that if $\kappa_{-1}=-$, then the negative literal $\neg z_{\ell-1}$ is in cl_ℓ and variable z_i is true, whereas cl_ℓ is not satisfied by $z_{\ell-1}$, either. Using the same argument, we can show that if all three jobs $V_{\ell-1,\kappa_{-1},1}$, $V_{\ell,\kappa_0,1}$ and $V_{\ell+1,\kappa_1,1}$ scheduled together with CL_ℓ are all false, then cl_ℓ is not satisfied by the assignment. Furthermore, according to our scheduling method, if we cannot schedule CL_ℓ^T with a true variable job, we schedule it with the false job $V^F_{\ell-1,\kappa_{-1},1}$. That means, among the three machines $\theta_{\text{V-C-D},\ell,i,\cdot}$, only $\theta_{\text{V-C-D},\ell-1,i,\cdot}$ is of kind (F,T,T) and has a load of $10^{14}\sigma_{max}+1$. The other two machines have a load of $10^{14}\sigma_{max}$. Similarly, we check variable-dummy machines and see that among the three machines $\theta_{\text{V-D},i,\cdot}$ where $i \in \{\ell-1,\ell,\ell+1\}$, only machine $\theta_{\text{V-D},\ell-1,\cdot}$ is of kind (T,T) and has a load of $10^{14}\sigma_{max}-1$. The other two machines have a load of $10^{14}\sigma_{max}$.

According to our observation in the above paragraph, we have the following claim.

CLAIM 3. *If $cl_\ell \in C_1$ is satisfied, then the three clause-variable-dummy machines $\theta_{\text{V-C-D},i,\ell,\cdot}$ and the three variable-dummy machines $\theta_{\text{V-D},i,\cdot}$, $i \in \{\ell-1,\ell,\ell+1\}$ all have a load of $10^{14}\sigma_{max}$; otherwise, machine $\theta_{\text{V-C-D},\ell-1,\ell,\cdot}$ has a load of $10^{14}\sigma_{max}+1$, $\theta_{\text{V-D},\ell-1,\cdot}$ has a load of $10^{14}\sigma_{max}-1$, and all the remaining 4 machines have a load of $10^{14}\sigma_{max}$.*

Combining Claim 2 amd Claim 3, we know that each unsatisfied clause can lead to at most 2 machines with load $10^{14}\sigma_{max}\pm 1$. Recall that the total processing time of all jobs is $10^{14}\sigma_{max}\cdot(2\gamma n+8n)$, hence the number of machines with load $10^{14}\sigma_{max}+1$ should equal the number of machines with load $10^{14}\sigma_{max}-1$. Consequently, if there are ϑn unsatisfied clauses, the resulted schedule will contain at most $2\vartheta n$ machines with load $10^{14}\sigma_{max}\pm 1$. Using Taylor's expression, we have that

$$\begin{aligned}(x+1)^q+(x-1)^q &= x^q[(1+\frac{1}{x})^q+(1-\frac{1}{x})^q]\\ &= x^q(1+\frac{q(q-1)}{2x^2}+o(\frac{1}{x^2}))\\ &= x^q+\frac{q(q-1)}{2}\cdot x^{q-2}+o(x^{q-2}),\end{aligned}$$

Hence, by simple calculations Lemma G.1 is proved.

G.3 Scheduling to 3SAT′ The goal of this subsection is to show that if the constructed scheduling instance admits a feasible schedule of a small objective value, then the given 3SAT′ instance admits a truth-assignment that satisfies most clauses. More precisely, we prove the following lemma.

LEMMA G.2. *If there are at least ϑn clauses not satisfied, then any feasible schedule has an objective value at least $(2\gamma n+8n)(10^{14}\sigma_{max})^q+\frac{q(q-1)\vartheta n}{48}\cdot(10^{14}\sigma_{max})^{q-2}+o(n\sigma_{max}^{q-2})$.*

In the following we consider a solution *Sol* for scheduling whose objective value is bounded by $(2\gamma n+8n)(10^{14}\sigma_{max})^q+(\frac{q(q-1)\vartheta n}{48}-\epsilon')\cdot(10^{14}\sigma_{max})^{q-2}$ for arbitrarily small $\epsilon'>0$.

Recall that we have constructed in total $2\gamma n+8n$ machines. According to Subsection G.2, the total processing time of all jobs is $(2\gamma n+8n)\cdot 10^{14}\sigma_{max}$. Consider an arbitrary schedule. We say a machine is good if its load is exactly $10^{14}\sigma_{max}$; otherwise, the machine is bad. Since the processing times are half-integral (multiples of 1/2), the load of a bad machine is either no larger than $10^{14}\sigma_{max}-0.5$, or no less than $10^{14}\sigma_{max}+0.5$. Furthermore, we say a machine is very bad if its load deviates from $10^{14}\sigma_{max}$ by at least σ_{max}, i.e., the load of a very bad machine is either no larger than $(10^{14}-1)\sigma_{max}$, or no smaller than $(10^{14}+1)\sigma_{max}$.

LEMMA G.3. *If there exists a very bad machine, then the objective value of the schedule is at least $m(10^{14}\sigma_{max})^q+c_1\sigma_{max}^q$ for some constant $c_1>0$.*

Towards the proof, we need the following lemma.

LEMMA G.4. *For $x, q, m > 1$ and $k \geq 1$, it holds that*

$$(x-k)^q + (m-1)(x + \frac{k}{m-1})^q \geq mx^q + \frac{q(q-1)}{4} \min\{(x-1)^{q-2}, (x+1)^{q-2}\}.$$

Proof. Taking the derivative of $(x-k)^q+(m-1)(x+\frac{k}{m-1})^q$ with respect to k, we get $-q(x-k)^{k-1}+q(x+\frac{k}{m-1})^{q-1} > 0$ when $x, q, m > 1$ and $k \geq 1$, hence the function $(x-k)^q + (m-1)(x+\frac{k}{m-1})^q$ is an increasing function of k, thus it suffices to prove the lemma for $k = 1$. According to the mean value theorem, we have

$$\begin{aligned}
\Gamma := \quad & \frac{1}{m}(x-1)^q + \frac{m-1}{m}(x + \frac{1}{m-1})^q - x^q \\
= \quad & -\frac{1}{m}[x^q - (x-1)^q] + \frac{m-1}{m}[(x + \frac{1}{m-1})^q - x^q] \\
= \quad & -\frac{1}{m}[x^q - (x-\frac{1}{2})^q] - \frac{1}{m}[(x-\frac{1}{2})^q - (x-1)^q] + \frac{m-1}{m}[(x + \frac{1}{m-1})^q - x^q] \\
= \quad & -\frac{1}{2m}q(x-\theta_1)^{q-1} - \frac{1}{2m}q(x-\frac{1}{2}-\theta_2)^{q-1} + \frac{m-1}{m} \cdot \frac{1}{m-1} \cdot q(x+\theta_3)^{q-1}
\end{aligned}$$

for some $\theta_1, \theta_2 \in (0, 1/2)$ and $\theta_3 \in (0, \frac{1}{m-1})$. Further apply the mean value theorem, we have

$$\begin{aligned}
\Gamma \quad \geq \quad & \frac{q}{2m}[(x+\theta_3)^{q-1} - (x-\frac{1}{2}-\theta_2)^{q-1}] \\
= \quad & \frac{q}{2m}(\theta_2 + \theta_3 + \frac{1}{2})(q-1)(x+\theta_4)^{q-2}
\end{aligned}$$

for some $\theta_4 \in (-1/2-\theta_2, \theta_3)$. If $q \geq 2$, then $(x+\theta_4)^{q-2} \geq (x-1)^{q-2}$. Otherwise $1 < q < 2$ and it holds that $(x+\theta_4)^{q-2} \geq (x+1)^{q-2}$. Thus

$$\Gamma \geq \frac{q(q-1)}{4m} \min\{(x-1)^{q-2}, (x+1)^{q-2}\}.$$

Hence, the lemma is proved. ∎

Similarly, we can prove that

LEMMA G.5. *For $x, q, m > 1$ and $k \geq 1$, it holds that*

$$(x+k)^q + (m-1)(x - \frac{k}{m-1})^q \geq mx^q + \frac{q(q-1)}{4} \min\{(x-1)^{q-2}, (x+1)^{q-2}\}.$$

Now we are ready to prove Lemma G.3.

Proof. [Proof of Lemma G.3] Suppose the load of one very bad machine is $(10^{14}-k)\sigma_{max}$ for some $|k| \geq 1$, then total load of all other machines is $10^{14}(m-1)\sigma_{max} + k\sigma_{max}$. By the convexity of the function x^q, the objective value of such a solution is at least:

$$\begin{aligned}
& (10^{14}-k)^q \sigma_{max}^q + (m-1) \cdot [\frac{10^{14}(m-1)\sigma_{max} + k\sigma_{max}}{m-1}]^q \\
= \quad & \sigma_{max}^q [(10^{14}+k)^q + (m-1)(10^{14} + \frac{k}{m-1})^q] \\
\geq \quad & m(10^{14}\sigma_{max})^q + \sigma_{max}^q \cdot \frac{q(q-1)}{4} \min\{(10^{14}-1)^{q-2}, (10^{14}+1)^{q-2}\}
\end{aligned}$$

Hence, the lemma is proved. ∎

We have shown that if a schedule admits a very bad machine, then its objective is significantly large and cannot be *Sol*. To prove Lemma G.2, it suffices to restrict our attention to schedules without any very bad machine.

Notice that the processing time of a gap job is at least $(10^{14} - 2 \times 10^{13})\sigma_{max}$, we know that there can be at most one gap job on a machine that is not very bad. Given the fact that the total number of gap jobs equals the number of machines, and there is no very bad machine in *Sol*, we have the following observation.

LEMMA G.6. *There is exactly one gap job on each machine in Sol.*

Given Lemma G.6, we will use the symbol of a gap job, e.g., $\theta_{\text{V-L},i,+}$, to denote the machine on which this job is scheduled.

The following lemma is straightforward by observing that $\sigma_{max} > x \cdot \sigma(i)$ for all i, and hence the type coordinates (i.e., the term $10^j\sigma_{max}$) of jobs on a machine that is not very bad cannot add up to smaller than $(10^{14} - 2)\sigma_{max}$ or larger than $(10^{14} + 2)\sigma_{max}$.

LEMMA G.7. *If in a solution there is no very bad machine, then*

- *On a variable-link machine $\theta_{V\text{-}L,i,\iota}$ where $\iota \in \{+, -\}$, there are exactly three jobs – a gap job, a variable job and a link job.*
- *On a link-link machine $\theta_{L\text{-}L,i,h,\iota}$ where $\iota \in \{+, -\}$, there are exactly three jobs – a gap job and two link jobs.*
- *On a variable-dummy machine $\theta_{V\text{-}D,i,\iota}$ where $\iota \in \{+, -\}$, there are exactly three jobs – a gap job, a variable job and a dummy job.*
- *On a variable-clause-dummy machine $\theta_{V\text{-}C\text{-}D,\ell,i,\iota}$ where $\iota \in \{+, -\}$, there are exactly four jobs – a gap job, a variable job, a clause job and a dummy job.*
- *On a variable-truth machine $\theta_{V\text{-}T,i,\rho}$ where $\rho \in \{(a, c), (b, d), (a, d), (b, c)\}$, there are exactly four jobs – a gap job, a variable job and two truth-assignment jobs; Furthermore, the two truth-assignment jobs are:*
  - $TR_{\cdot,a}$ *and* $TR_{\cdot,c}$ *if* $\rho = (a, c)$;
  - $TR_{\cdot,b}$ *and* $TR_{\cdot,d}$ *if* $\rho = (b, d)$;
  - $TR_{\cdot,a}$ *and* $TR_{\cdot,d}$ *if* $\rho = (a, d)$;
  - $TR_{\cdot,b}$ *and* $TR_{\cdot,c}$ *if* $\rho = (b, c)$.

Proof. The proof can be carried out through a counting argument in the order of dummy jobs, clause jobs, truth-assignment jobs, link jobs and variable jobs according to Table 1. In the following, we prove dummy jobs and the other types of jobs can be proved in a similar way. The reader may refer to Table 2 for a quick overview on job processing times. Note that a dummy job has a processing time at least $(10^{13} - 1/2)\sigma_{max}$. It is easy to see that if a variable-link machine, or link-link machine, or variable-truth machine accepts one dummy job, then the load of this machine is larger than $(10^{14} + 1)\sigma_{max}$, contradicting the fact that there is no very bad machine. Hence, dummy jobs can only be scheduled on variable-clause-dummy machine or a variable-dummy machine. Similarly, if a variable-clause-dummy machine or a variable-dummy machine accepts two or more dummy jobs, its load becomes larger than $(10^{14} + 1)\sigma_{max}$, hence each of these machines can accept at most 1 dummy job. On the other hand, there are $2n$ dummy jobs, which is equal to the sum of the number of variable-clause-dummy machines (which is n) and the number of variable-dummy machines (which is also n). Hence, each variable-clause-dummy machine or variable-dummy machine accepts exactly one dummy job. Subtracting one dummy job together with the gap job on each variable-clause-dummy machine or variable-dummy machine, we know that if the machine is not very bad, then the remaining jobs on a variable-clause-dummy machine should add up to some value within $[(10^{12} + 10^5 - 2)\sigma_{max}, (10^{12} + 10^5 + 2)\sigma_{max}]$ (if this machine is $\theta_{\text{V-C-D},\ell,i,+}$) or $[(10^{12} + 10^3 - 2)\sigma_{max}, (10^{12} + 10^3 + 2)\sigma_{max}]$ (if this machine is $\theta_{\text{V-C-D},\ell,i,-}$), and the remaining jobs on a variable-dummy machine should add up to some value within $[(10^5 - 2)\sigma_{max}, (10^5 + 2)\sigma_{max}]$ (if this machine is $\theta_{\text{V-D},i,+}$) or $[(10^3 - 2)\sigma_{max}, (10^3 + 2)\sigma_{max}]$ (if this machine is $\theta_{\text{V-D},i,-}$). Consequently, we can apply the same argument to clause jobs, and then truth-assignment jobs, then link jobs and then variable jobs. ∎

Using Lemma G.7, we further have the following observation.

LEMMA G.8. *On a good machine, the type-component of jobs add up to $10^{14}\sigma_{max}$, the index-components and the true/false-components of jobs add up to 0, respectively.*

Now we further identify the index-component of jobs on each machine.

LEMMA G.9. *Consider an arbitrary variable-dummy machine $\theta_{V\text{-}D,i,\iota}$ where $\iota \in \{+,-\}$. If the machine is good, then the variable job on this machine is $V_{i,\iota,2}$.*

Applying Lemma G.8, the proof is straightforward by checking the sum of type-components and index-components of jobs, respectively.

LEMMA G.10. *Consider an arbitrary variable-clause-dummy machine $\theta_{V\text{-}C\text{-}D,\ell,i,\iota}$ where $\iota \in \{+,-\}$. If the machine is good, then the clause job on this machine is CL_ℓ, and the variable job on this machine is $V_{i,\iota,1}$.*

Proof. By Lemma G.8, the type-components of the three jobs add up to $10^{14}\sigma_{max}$, hence it is easy to see that the variable job should be $V_{i',\iota,1}$ for some i'. Let the clause job be $\mathrm{CL}_{\ell'}$ for some ℓ'. As the index-components of the three jobs add up to 0, we have

$$\sigma(\ell') + \sigma(i') = \sigma(\ell) + \sigma(i).$$

Notice that for any machine $\theta_{\text{V-C-D},\ell,i,\iota}$ it holds that $i \in \{\ell-1,\ell,\ell+1\}$ and $\ell \in \{2,5,\cdots,n-1\}$. We claim that $\ell' = \ell$ and $i' = i$. To see why, consider two cases. If $\ell = i$, then $\sigma(\ell') + \sigma(i') = 2\sigma(\ell)$. According to Lemma 3.3, we have $\ell' = i' = \ell = i$ and the claim follows. Otherwise, $i = \ell \pm 1$. According to Lemma 3.3, the only solution for $\sigma(j) + \sigma(j+1) = \sum_{h=1}^{k} \sigma(j_h)$, $k \le 5$, is $k = 2$ and $\{j_1, j_2\} = \{j, j+1\}$. Hence, we have $\{\ell', i'\} = \{\ell, i\}$. Note that $\ell', \ell \in \{2,5,\cdots,n-1\}$, hence $\ell', \ell \equiv 2(\mod 3)$. But $i \not\equiv 2(\mod 3)$. Thus, $\ell = \ell'$ and $i = i'$. In both cases, Lemma G.10 holds. ∎

LEMMA G.11. *Consider an arbitrary variable-truth machine $\theta_{V\text{-}T,i,\rho}$ where $\rho \in \{(a,c),(b,d),(a,d),(b,c)\}$. If the machine is good, then the variable and truth-assignment jobs are:*

- *$V_{i,+,1}$ and $TR_{i,a}$, $TR_{i,c}$ if $\rho = (a,c)$;*
- *$V_{i,+,2}$ and $TR_{i,b}$, $TR_{i,d}$ if $\rho = (b,d)$;*
- *$V_{i,-,1}$ and $TR_{i,a}$, $TR_{i,d}$ if $\rho = (a,d)$;*
- *$V_{i,-,2}$ and $TR_{i,b}$, $TR_{i,c}$ if $\rho = (b,c)$.*

Proof. According to Lemma G.8, the type-components of jobs add up to $10^{14}\sigma_{max}$. Hence, it is easy to verify that for some i_1, i_2, i_3 the variable and truth-assignment jobs are $V_{i_1,+,1}$ and $TR_{i_2,a}$, $TR_{i_3,c}$, if $\rho = (a,c)$; $V_{i_1,+,2}$ and $TR_{i_2,b}$, $TR_{i_3,d}$, if $\rho = (b,d)$; $V_{i_1,-,1}$ and $TR_{i_2,a}$, $TR_{i_3,d}$, if $\rho = (a,d)$; $V_{i_1,-,2}$ and $TR_{i_2,b}$, $TR_{i_3,c}$, if $\rho = (b,c)$.

We prove $i_1 = i_2 = i_3 = i$, and Lemma G.11 follows. According to Lemma G.8, the index-components add up to 0, hence

$$10\sigma(i_1) + 10\sigma(i_2) + 10\sigma(i_3) = 30\sigma(i),$$

i.e., $\sigma(i_1) + \sigma(i_2) + \sigma(i_3) = 3\sigma(i)$. According to Lemma 3.3, the above equation has a unique solution, which is $i_1 = i_2 = i_3 = i$. ∎

LEMMA G.12. *Consider an arbitrary variable-link machine $\theta_{V\text{-}L,i,\iota}$ where $\iota \in \{+,-\}$. If the machine is good, then the variable job on this machine is $V_{i,\iota,2}$, and the link job on this machine is $LN_{i,1,\iota}$.*

Proof. Using the fact that the type-components of all jobs add up to $10^{14}\sigma_{max}$, it is easy to see that the variable job should be $V_{i_1,\iota,2}$ and the link job should be $\mathrm{LN}_{i_2,h,\iota}$ for some $1 \le i_1, i_2 \le n$ and $1 \le h \le 2\gamma+2$. We prove the lemma for $\iota = +$. The case that $\iota = -$ can be proved in the same way.

Given that the index-components should add up to 0, we have the following:

$$\mathbf{a}_i + \mathbf{b}_i^1 = \mathbf{a}_{i_1} + \mathbf{b}_{i_2}^h \tag{G.6}$$

Recall that $\mathbf{a}_i = \mathbf{b}_i^0$. According to Lemma 3.7, we have $h = 1$ and $i_1 = i_2 = i$. ∎

LEMMA G.13. *Consider an arbitrary link-link machine $\theta_{L\text{-}L,i,h,\iota}$ where $\iota \in \{+,-\}$, $1 \le h \le 2\gamma+1$. If the machine is good, then the two link jobs on this machine are $LN_{i,h,\iota}$ and $LN_{i,h+1,\iota}$.*

The proof is similar to that of Lemma G.12 by utilizing Lemma 3.7.

LEMMA G.14. *Consider an arbitrary link-link machine $\theta_{L\text{-}L,i,+,-}$. If the machine is good, then the two link jobs on this machine are $LN_{i,2\gamma+2,+}$ and $LN_{\tau(i),2\gamma+2,-}$.*

Proof. Using the fact that the type-components of all jobs add up to $10^{14}\sigma_{max}$, it is easy to see that the two link jobs should $\mathrm{LN}_{i_1,h_1,+}$ and $\mathrm{LN}_{i_2,h_2,-}$. Given that the index-components should add up to 0, we have the following:

$$2\mathbf{b}_i^{2\gamma+2} = \mathbf{b}_{i_1}^{h_1} + \hat{\mathbf{b}}_{i_2}^{h_2} \tag{G.7}$$

According to Lemma 3.8, we have $i_1 = i$ and $i_2 = \tau(i)$, and $h_1 = h_2 = 2\gamma + 2$. ∎

We have proved, so far, that if a machine is good, then the jobs scheduled on it must follow Table 3. Finally we consider the true/false-components of jobs on good machines. Based on the T/F-component of jobs, the following lemma is easy to verify.

LEMMA G.15. *The followings are true:*

- *If a variable-link machine is good, then the T/F-type of the variable job and link job on this machine is (T,F) or (F,T);*
- *If a link-link machine is good, then the T/F-type of the two link jobs on this machine is (T,F) or (F,T);*
- *If a variable-clause-dummy machine is good, then the T/F-type of the variable job, clause job and dummy job on this machine is (T,T,T) or (T,F,F) or (F,F,T);*
- *If a variable-dummy machine is good, then the T/F-type of the variable job and dummy job on this machine is (T,F) or (F,T);*
- *If a variable-truth machine is good, then the T/F-type of the variable job and two truth-assignment jobs on this machine is (T,T,T) or (F,F,F);*

G.3.1 Truth-assignment based on scheduling Given a feasible schedule *Sol*, we give a truth-assignment of I_{sat} as follows: if the job $V_{i,+,1}^T$ is scheduled on machine $\theta_{\text{V-T},i,a,c}$, then we let variable z_i be false; if the job $V_{i,+,1}^F$ is scheduled on machine $\theta_{\text{V-T},i,a,c}$, then we let variable z_i be true. If $V_{i,+,1}$ is not scheduled on machine $\theta_{\text{V-T},i,a,c}$, we let z_i be true.

We call the machines in the following Table 6 as machines of group i. Notice that groups are not disjoint, particularly machines $\theta_{\text{L-L},i,+,-}$, $\theta_{\text{L-L},\tau^{-1}(i),+,-}$ will appear in two groups. Besides the two machines, all other machines in a group do not appear in other groups. We have the following lemma.

LEMMA G.16. *Suppose all machines in group i are good. If $V_{i,+,1}^F$ is scheduled on machine $\theta_{V\text{-}T,i,a,c}$, then the jobs scheduled on these machines are according to Table 6 ; if $V_{i,+,1}^T$ is scheduled on machine $\theta_{V\text{-}T,i,a,c}$, then the jobs scheduled on these machines are according to Table 7.*

Proof. We prove the first half of Lemma G.16, the second half can be proved in the same way. If $V_{i,+,1}^T$ is scheduled on machine $\theta_{\text{V-T},i,a,c}$, then by Lemma G.15 we know the other two jobs are $\mathrm{TR}_{i,a}^F$ and $\mathrm{TR}_{i,c}^F$, consequently, $\mathrm{TR}_{i,a}^T$ is scheduled on machine $\theta_{\text{V-T},i,a,d}$. Using similar argument it is easy to see the jobs scheduled on the 4 variable-truth machines follow Table 6.

We consider variable-clause-dummy and variable-dummy machines. It follows that the remaining $V_{i,+,1}^T$ and $V_{i,-,1}^F$ are scheduled on these machines. The T/F type of the other jobs on these machines follow from Lemma G.15.

Next, we consider variable-link machines. Again by the scheduling on variable-truth machines, the remaining $V_{i,+,2}^T$ and $V_{i,-,2}^F$ are scheduled on these machines. The T/F-type of the link jobs are determined by Lemma G.15.

Finally we consider link-link machines. Based on the link jobs scheduled on variable-link machines and Lemma G.15, $\mathrm{LN}_{i,1,+}^F$ and $\mathrm{LN}_{i,2,+}^T$ must be scheduled on $\theta_{\text{L-L},i,1,+}$, consequently the remaining $\mathrm{LN}_{i,2,+}^F$ must be

Variable-Link	$\theta_{\text{V-L},i,+}$	$V^T_{i,+,2}$	$\text{LN}^F_{i,1,+}$	\
	$\theta_{\text{V-L},i,-}$	$V^F_{i,-,2}$	$\text{LN}^T_{i,1,-}$	\
Link-Link $h \in \{1,2,\cdots,2\gamma+1\}$	$\theta_{\text{L-L},i,h,+}$	$\text{LN}^F_{i,h,+}$	$\text{LN}^T_{i,h+1,+}$	\
	$\theta_{\text{L-L},i,h,-}$	$\text{LN}^F_{i,h,-}$	$\text{LN}^T_{i,h+1,-}$	\
	$\theta_{\text{L-L},i,+,-}$	$\text{LN}^T_{i,2\gamma+2,+}$	$\text{LN}^F_{\tau(i),2\gamma+2,-}$	\
	$\theta_{\text{L-L},\tau^{-1}(i),+,-}$	$\text{LN}^T_{\tau^{-1}(i),2\gamma+2,+}$	$\text{LN}^F_{i,2\gamma+2,-}$	\
Variable-Clause-Dummy & Variable-Dummy Case 1: positive literal $z_i \in C_1$	$\theta_{\text{V-C-D},\ell,i,+}$	$V^T_{i,+,1}$	CL^*_ℓ	DM^*
	$\theta_{\text{V-D},i,-}$	$V^F_{i,-,1}$	DM^T	\
Variable-Clause-Dummy & Variable-Dummy Case 2: negative literal $\neg z_i \in C_1$	$\theta_{\text{V-C-D},i,-}$	$V^F_{i,-,1}$	CL^F_ℓ	DM^T
	$\theta_{\text{V-D},i,+}$	$V^T_{i,+,1}$	DM^F	\
Variable-Truth	$\theta_{\text{V-T},i,a,c}$	$V^F_{i,+,1}$	$\text{TR}^F_{i,a}$	$\text{TR}^F_{i,c}$
	$\theta_{\text{V-T},i,b,d}$	$V^F_{i,+,2}$	$\text{TR}^F_{i,b}$	$\text{TR}^F_{i,d}$
	$\theta_{\text{V-T},i,a,d}$	$V^T_{i,-,1}$	$\text{TR}^T_{i,a}$	$\text{TR}^T_{i,d}$
	$\theta_{\text{V-T},i,b,c}$	$V^T_{i,-,2}$	$\text{TR}^T_{i,b}$	$\text{TR}^T_{i,c}$

Table 6: Scheduling of group i machines when $V^F_{i,+,1}$ is scheduled on machine $\theta_{\text{V-T},i,a,c}$ (and we set variable z_i to be true)

Variable-Link	$\theta_{\text{V-L},i,+}$	$V^F_{i,+,2}$	$\text{LN}^T_{i,1,+}$	\
	$\theta_{\text{V-L},i,-}$	$V^T_{i,-,2}$	$\text{LN}^F_{i,1,-}$	\
Link-Link $h \in \{1,2,\cdots,2\gamma+1\}$	$\theta_{\text{L-L},i,h,+}$	$\text{LN}^T_{i,h,+}$	$\text{LN}^F_{i,h+1,+}$	\
	$\theta_{\text{L-L},i,h,-}$	$\text{LN}^T_{i,h,-}$	$\text{LN}^F_{i,h+1,-}$	\
	$\theta_{\text{L-L},i,+,-}$	$\text{LN}^F_{i,2\gamma+2,+}$	$\text{LN}^T_{\tau(i),2\gamma+2,-}$	\
	$\theta_{\text{L-L},\tau^{-1}(i),+,-}$	$\text{LN}^F_{\tau^{-1}(i),2\gamma+2,+}$	$\text{LN}^T_{i,2\gamma+2,-}$	\
Variable-Clause-Dummy & Variable-Dummy Case 1: positive literal $z_i \in C_1$	$\theta_{\text{V-C-D},\ell,i,+}$	$V^F_{i,+,1}$	CL^F_ℓ	DM^T
	$\theta_{\text{V-D},i,-}$	$V^T_{i,-,1}$	DM^F	\
Variable-Clause-Dummy & Variable-Dummy Case 2: negative literal $\neg z_i \in C_1$	$\theta_{\text{V-C-D},i,-}$	$V^T_{i,-,1}$	CL^*_ℓ	DM^*
	$\theta_{\text{V-D},i,+}$	$V^F_{i,+,1}$	DM^T	\
Variable-Truth	$\theta_{\text{V-T},i,a,c}$	$V^T_{i,+,1}$	$\text{TR}^T_{i,a}$	$\text{TR}^T_{i,c}$
	$\theta_{\text{V-T},i,b,d}$	$V^T_{i,+,2}$	$\text{TR}^T_{i,b}$	$\text{TR}^T_{i,d}$
	$\theta_{\text{V-T},i,a,d}$	$V^F_{i,-,1}$	$\text{TR}^F_{i,a}$	$\text{TR}^F_{i,d}$
	$\theta_{\text{V-T},i,b,c}$	$V^F_{i,-,2}$	$\text{TR}^F_{i,b}$	$\text{TR}^F_{i,c}$

Table 7: Scheduling of group i machines when $V^T_{i,+,1}$ is scheduled on machine $\theta_{\text{V-T},i,a,c}$ (and we set variable z_i to be false)

scheduled on machine $\theta_{\text{L-L},i,2,+}$. Iteratively carrying on the above argument we can show that jobs scheduled on machines $\theta_{\text{L-L},i,h,+}$ must follow Table 6. Similar arguments can be applied to machines $\theta_{\text{L-L},i,h,-}$, $\theta_{\text{L-L},i,+,-}$ and $\theta_{\text{L-L},\tau^{-1}(i),+,-}$. □

The T/F-type of the clause job CL_ℓ is not determined in Table 6. Recall that among three copies of CL_ℓ there is one true copy CL^T_ℓ. Suppose CL^T_ℓ is scheduled on group i machines. If $V^T_{i,+,1}$ is scheduled on machine $\theta_{\text{V-T},i,a,c}$ and we set variable z_i to be true, then from Table 6 we know case 1 must happen, which implies that clause cl_ℓ is satisfied by z_i. If $V^F_{i,+,1}$ is scheduled on machine $\theta_{\text{V-T},i,a,c}$ and we set variable z_i to be false, then from Table 7 we know case 2 must happen, which implies that clause cl_ℓ is satisfied by $\neg z_i$. Hence the following lemma is true.

LEMMA G.17. *If all machines in group i are good and CL_ℓ^T is scheduled on these machines, then the clause $cl_\ell \in C_1$ that contains variable z_i is satisfied by this variable.*

Now consider clauses in C_2 and we have the following lemma.

LEMMA G.18. *If all machines in group i are good, and all machines in group $\tau(i)$ are also good, then the clause $(z_i \oplus \neg z_{\tau(i)})$ is satisfied.*

Proof. There are two possibilities. If $V_{i,+,1}^F$ is scheduled on machine $\theta_{\text{V-T},i,a,c}$ and we set variable z_i to be true, then $\text{LN}_{\tau(i),2\gamma+2,-}^F$, implying that $\text{LN}_{\tau(i),2\gamma+2,-}^T$ is scheduled in group $\tau(i)$. By checking Table 6 and Table 7 for variable $z_{\tau(i)}$, it follows that Table 6 is the case when $\text{LN}_{\tau(i),2\gamma+2,-}^T$ is scheduled, and consequently variable $z_{\tau(i)}$ is set to be true, whereas $(z_i \oplus \neg z_{\tau(i)})$ is satisfied. The other case when $V_{i,+,1}^T$ is scheduled on machine $\theta_{\text{V-T},i,a,c}$ and we set variable z_i to be false can be proved in a similar way. ∎

LEMMA G.19. *In a feasible schedule Sol, if there are at most m' machines which are not good, then in the corresponding truth-assignment, there are at most $6m'$ clauses that are not satisfied.*

Proof. We say a group is good if all machines in this group are good. According to Lemma G.17, if a clause cl_ℓ in C_1 is not satisfied, then the group that contains the job CL_ℓ^T is not good, that is, there is at least one machine that is not good in this group. Hence, if there are m_1 clauses in C_1 not satisfied, then there are at least m_1 groups that are not good. According to Lemma G.18, if a clause $(z_i \oplus \neg z_{\tau(i)})$ in C_2 is not good, then among group i and group $\tau(i)$ there is at least one group which is not good. Given that each group is only involved in two clauses of C_2, if there are m_2 clauses in C_2 not satisfied, then there are at least $m_2/2$ groups which are not good. Hence, there are at least $\max\{m_1, m_2/2\}$ groups which are not good, given $m_1 + m_2$ clauses which are not satisfied. Using the fact that $\frac{m_1+m_2}{\max\{m_1,m_2/2\}} \leq 3$, we know if there are at most m' machines which are not good, then there are at most $2m'$ groups which are not good, and hence there are at most $6m'$ clauses which are not satisfied. ∎

LEMMA G.20. *In a feasible schedule, if there are at least m' machines which are not good, then its objective value is at least $(2\gamma n + 8n) \times (10^{14}\sigma_{max})^2 + m'/4$.*

Proof. Recall that if a machine is not good, then its load is either $\geq 10^{14}\sigma_{max} + 0.5$ or $\leq 10^{14}\sigma_{max} - 0.5$. Suppose there are m_1' machines, with load $10^{14}\sigma_{max} + \mu_1$, $10^{14}\sigma_{max} + \mu_2$, $\cdots$, $10^{14}\sigma_{max} + \mu_{m_1'}$ where $\mu_j \geq 1/2$; there are m_2' machines, with load $10^{14}\sigma_{max} - \nu_1$, $10^{14}\sigma_{max} - \nu_2$, $\cdots$, $10^{14}\sigma_{max} + \nu_{m_2'}$ where $\nu_j \geq 1/2$. It follows that $\sum_j \mu_j = \sum_j \nu_j$ and $m_1' + m_2' = m'$. The objective value of the schedule is

$$
\begin{aligned}
& (2\gamma n + 8n - m')(10^{14}\sigma_{max})^q + \sum_{j=1}^{m_1'}(10^{14}\sigma_{max} + \mu_j)^q + \sum_{j=1}^{m_2'}(10^{14}\sigma_{max} - \nu_j)^q \\
= \; & (2\gamma n + 8n)(10^{14}\sigma_{max})^q + \frac{q(q-1)}{2}[\sum_{j=1}^{m_1'}\mu_j^2 + \sum_{j=1}^{m_2'}\nu_j^2](10^{14}\sigma_{max})^{q-2} + o([\sum_{j=1}^{m_1'}\mu_j^2 + \sum_{j=1}^{m_2'}\nu_j^2]\sigma_{max}^{q-2}) \\
\geq \; & (2\gamma n + 8n)(10^{14}\sigma_{max})^q + \frac{q(q-1)}{2}[\frac{m_1'}{4} + \frac{m_2'}{4}](10^{14}\sigma_{max})^{q-2} + o(m'\sigma_{max}^{q-2}) \\
= \; & (2\gamma n + 8n)(10^{14}\sigma_{max})^q + \frac{q(q-1)m'}{8}(10^{14}\sigma_{max})^{q-2} + o(m'\sigma_{max}^{q-2}).
\end{aligned}
$$

∎

Combining the above lemmas, Lemma G.2 is proved.

G.4 Finalizing the Proof of Theorem 3.1 Suppose on the contrary there exists a PTAS for $P||\sum_i C_i^q$ that runs in time $2^{O((1/\varepsilon)^{1/2-\delta})} + n^{O(1)}$, we show that this algorithm can be used to distinguish between instances of 3SAT′ with $4n/3$ clauses where at least $4(1-\epsilon')/3$ clauses are satisfiable from instances where at most $4(\beta+\epsilon')n/3$ clauses are satisfiable in time $2^{O(n^{1-\delta})}$, contradicting Lemma 3.1.

Consider the constructed scheduling instance with $2\gamma n + 8n = O(\frac{n \log n}{\log \log n})$ machines. Recall $\sigma_{max} = n^{1+O(\frac{1}{\log \log n})}$. If the 3SAT$'$ instance has at most $4\epsilon' n/3$ unsatisfied clauses, then by Lemma G.1 (taking $\vartheta = 4\epsilon'/3$) the objective value Obj_1 of the constructed scheduling instance is at most

$$\begin{aligned} Obj_1 &\leq (2\gamma n + 8n)(10^{14}\sigma_{max})^q + 4\epsilon' n/3 \cdot \frac{q(q-1)}{2}(10^{14}\sigma_{max})^{q-2} + o(n\sigma_{max}^{q-2}) \\ &= (2\gamma n + 8n)(10^{14}\sigma_{max})^q + \frac{2\epsilon' q(q-1)n}{3} \cdot (10^{14}\sigma_{max})^{q-2} + o(n\sigma_{max}^{q-2}) \end{aligned}$$

If the 3SAT$'$ instance has at least $4(1-\beta-\epsilon')n/3$ unsatisfied clauses, then by Lemma G.2 (taking $\vartheta = 4(1-\beta-\epsilon')/3$) the objective value Obj_2 of any feasible solution for the constructed scheduling instance is at least

$$\begin{aligned} Obj_2 &\geq (2\gamma n + 8n)(10^{14}\sigma_{max})^q + \frac{q(q-1) \cdot 4(1-\beta-\epsilon')n/3}{48} \cdot (10^{14}\sigma_{max})^{q-2} + o(n\sigma_{max}^{q-2}) \\ &= (2\gamma n + 8n)(10^{14}\sigma_{max})^q + \frac{q(q-1)(1-\beta-\epsilon')n}{36} \cdot (10^{14}\sigma_{max})^{q-2} + o(n\sigma_{max}^{q-2}) \end{aligned}$$

for some constant $\beta < 1$.

We apply the PTAS for $P||\sum_i C_i^q$ by setting $\varepsilon = \frac{1}{(2\gamma+8)\times(10^{14}\sigma_{max})^2} \cdot \frac{q(q-1)\epsilon'}{36} = \Theta(\gamma^{-1}\sigma_{max}^{-2}) = n^{-2-O(\frac{\log \log n}{\log n})}$, then it follows that the PTAS runs in time $2^{O(n^{1-o(1)})}$. If there exists a feasible schedule with objective value at most Obj_1, then the PTAS returns a solution with objective value at most

$$Obj_1 \cdot (1+\varepsilon) \leq (2\gamma n + 8n)(10^{14}\sigma_{max})^q + \frac{q(q-1)\epsilon' n}{36} \cdot (10^{14}\sigma_{max})^{q-2} + o(n\sigma_{max}^{q-2}) < Obj_2.$$

Otherwise, any feasible solution has an objective value of at least Obj_2. That is, the PTAS can be used to distinguish between scheduling instances that admit a feasible schedule at most Obj_1 and scheduling instances that do not admit any feasible schedule of objective value no more than Obj_2, and thus can also be used to distinguish 3SAT$'$ where at least $4(1-\epsilon')n/3$ clauses are satisfiable from instances where at most $4(\beta+\epsilon')n/3$ clauses are satisfiable, contradicting Lemma 3.1.

Remark. It is important to observe that our reduction is only valid when the number of machines $m = \tilde{O}(n) = \tilde{O}(\sqrt{1/\epsilon})$. If $m = O((1/\epsilon)^\kappa)$ for $\kappa < 1/2$, then applying the same reduction we have $\epsilon = \tilde{O}(n^{-1/\kappa})$ by using that $m = \tilde{O}(n)$, whereas we have Corollary G.1 below. On the other hand, if $m = \Omega((1/\epsilon)^\kappa)$ for $\kappa > 1/2$, then we also have $n = \Omega((1/\epsilon)^\kappa)$. The objective value is $\Theta(mT^2) = \Omega((1/\epsilon)^{3\kappa})$. Therefore, a PTAS brings an error of $O(mT^2\epsilon) = \Omega(n \cdot (1/\epsilon)^{2\kappa-1}) = \Omega(n)$, which is large enough to accommodate the gap of $O(n)$ in the reduction, i.e., the reduction does not work any more.

COROLLARY G.1. *Let $q > 1$ be an arbitrary constant. Assuming ETH, for any ϵ such that $m = O((1/\epsilon)^\kappa)$ for some $\kappa \leq 1/2$, there is no $(1+\epsilon)$-approximation algorithm for $P||\sum_i C_i^q$ that runs in time $2^{O((1/\varepsilon)^{\kappa-\delta})} + n^{O(1)}$ time for any constant $\delta > 0$.*

References

[1] N. Alon, Y. Azar, G.J. Woeginger, and T. Yadid. Approximation schemes for scheduling. In *Proceedings of the 8th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 493–500, 1997.

[2] N. Alon, Y. Azar, G.J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.

[3] F.A. Behrend. On sets of integers which contain no three terms in arithmetical progression. *Proceedings of the National Academy of Sciences of the United States of America*, 32(12):331, 1946.

[4] P.R. Berman, A.D. Scott, and M. Karpinski. Approximation hardness and satisfiability of bounded occurrence instances of sat. Technical report, SIS-2003-269, 2003.

[5] E. Bonnet, B. Escoffier, E.J. Kim, and V.T. Paschos. On subexponential and fpt-time inapproximability. *Algorithmica*, 71(3):541–565, 2015.

[6] J. Chen, X. Huang, I.A. Kanj, and G. Xia. Linear fpt reductions and computational lower bounds. In *Proceedings of the 36th annual ACM Symposium on Theory of Computing*, pages 212–221, 2004.

[7] L. Chen, K. Jansen, and G. Zhang. On the optimality of approximation schemes for the classical scheduling problem. In *Proceedings of the 25th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 657–668, 2014.

[8] L. Chen, D. Marx, D. Ye, and G. Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In *Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science*, pages 22:1–14, 2017.

[9] E.D. Demaine, F.V. Fomin, M. Hajiaghayi, and D.M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. *Journal of the ACM*, 52(6):866–893, 2005.

[10] P. Erdös and P. Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, 1(4):212–215, 1941.

[11] M.R. Garey and D.S. Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[12] W. Gasarch, J. Glenn, and C.P. Kruskal. Finding large 3-free sets i: The small n case. *Journal of Computer and System Sciences*, 74(4):628–655, 2008.

[13] 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 Mathematics*, 5:287–326, 1979.

[14] D.S. Hochbaum. Various notions of approximations: Good, better, best and more. *Approximation Algorithms for NP-Hard Problems*, 1997.

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

[16] S. Ibrahimpur and C. Swamy. Minimum-norm load balancing is (almost) as easy as minimizing makespan. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming*, pages 81:1–81:20, 2021.

[17] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[18] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

[19] K. Jansen. An eptas for scheduling jobs on uniform processors: using an milp relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, 24(2):457–485, 2010.

[20] K. Jansen, K. Klein, and J. Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Mathematics of Operations Research*, 45(4):1371–1392, 2020.

[21] K. Jansen, F. Land, and K. Land. Bounding the running time of algorithms for scheduling and packing problems. *SIAM Journal on Discrete Mathematics*, 30(1):343–366, 2016.

[22] K. Jansen, M. Maack, and R. Solis-Oba. Structural parameters for scheduling with assignment restrictions. *Theoretical Computer Science*, 844:154–170, 2020.

[23] N. Karmarkar and R.M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd annual Symposium on Foundations of Computer Science*, pages 312–320, 1982.

[24] P.N. Klein and D. Marx. Solving planar k-terminal cut in time. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming*, pages 569–580, 2012.

[25] D. Knop and M. Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21(5):493–503, 2018.

[26] D. König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen*, 77(4):453–465, 1916.

[27] J. Leung. Bin packing with restricted piece sizes. *Information Processing Letters*, 31(3):145–149, 1989.

[28] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of EATCS*, 3(105), 2013.

[29] D. Marx. On the optimality of planar and geometric approximation schemes. In *Proceedings of the 48th annual IEEE Symposium on Foundations of Computer Science*, pages 338–348, 2007.
[30] D. Marx. A tight lower bound for planar multiway cut with fixed number of terminals. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming*, pages 677–688, 2012.
[31] M. Mnich and R. van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100:254–261, 2018.
[32] M. Mnich and A. Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1):533–562, 2015.
[33] L. Moser. On non-averaging sets of integers. *Canadian Journal of Mathematics*, 5:245–252, 1953.
[34] D. Moshkovitz and R. Raz. Two-query pcp with subconstant error. *Journal of the ACM*, 57(5):29, 2010.
[35] K. O'Bryant. A complete annotated bibliography of work related to sidon sequences. *arXiv preprint math/0407117*, 2004.
[36] M. Pilipczuk, M. Pilipczuk, P. Sankowski, and E.J. van Leeuwen. Network sparsification for steiner problems on planar and bounded-genus graphs. *ACM Transactions on Algorithms*, 14(4):1–73, 2018.
[37] M. Skutella and G.J. Woeginger. A ptas for minimizing the weighted sum of job completion times on parallel machines. In *Proceedings of the 31st annual ACM Symposium on Theory of Computing*, pages 400–407, 1999.
[38] C. Tovey. A simplified satisfiability problem. *Disc. Appl. Math.*, 8:85–89, 1984.
[39] L. Trevisan. Inapproximability of combinatorial optimization problems. *arXiv preprint cs/0409043*, 2004.
[40] D.P. Williamson and D.B. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.
[41] G.J. Woeginger. When does a dynamic programming formulation guarantee the existence of an fptas? In *Proceedings of the 10th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 820–829, 1999.