Original citation:
Cygan, Marek, Pilipczuk, Marcin, Pilipczuk, Michał and Wojtaszczyk, Jakub Onufry. (2014) Scheduling partially ordered jobs faster than 2 n. Algorithmica, Volume 68 (Number 3). pp. 692-714.

Original citation:

Cygan, Marek, Pilipczuk, Marcin, Pilipczuk, Michał and Wojtaszczyk, Jakub Onufry. (2014) Scheduling partially ordered jobs faster than 2 n. Algorithmica, Volume 68 (Number 3). pp. 692-714.

Algorithmica (2014) 68:692–714
DOI 10.1007/s00453-012-9694-7

Scheduling Partially Ordered Jobs Faster than 2^n

Marek Cygan · Marcin Pilipczuk ·
Michał Pilipczuk · Jakub Onufry Wojtaszczyk

Received: 20 December 2011 / Accepted: 24 September 2012 / Published online: 9 October 2012

Abstract In a scheduling problem, denoted by $1|\text{prec}|\sum C_i$ in the Graham notation, we are given a set of n jobs, together with their processing times and precedence constraints. The task is to order the jobs so that their total completion time is minimized. $1|\text{prec}|\sum C_i$ is a special case of the Traveling Repairman Problem with precedences. A natural dynamic programming algorithm solves both these problems in $2^n n^{O(1)}$ time, and whether there exists an algorithms solving $1|\text{prec}|\sum C_i$ in $O(c^n)$ time for some constant $c < 2$ was an open problem posted in 2004 by Woeginger. In this paper we answer this question positively.

Keywords Moderately-exponential algorithms · Dynamic programming · 2^n-Barrier · Scheduling · Partially ordered jobs

---

An extended abstract of this paper appears at 19th European Symposium on Algorithms, Saarbrücken, Germany, 2011.

M. Cygan and Marcin Pilipczuk were supported by Polish Ministry of Science grant no. N206 355636 and Foundation for Polish Science.

M. Cygan · M. Pilipczuk (✉)
Institute of Informatics, University of Warsaw, Warsaw, Poland
e-mail: malcin@mimuw.edu.pl

M. Cygan
e-mail: cygan@mimuw.edu.pl

M. Pilipczuk
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland
e-mail: michal.pilipczuk@students.mimuw.edu.pl

J.O. Wojtaszczyk
Google Inc., Cracow, Poland
e-mail: onufry@google.com

1 Introduction

It is commonly believed that no NP-hard problem is solvable in polynomial time. However, while all NP-complete problems are equivalent with respect to polynomial time reductions, they appear to be very different with respect to the best exponential time exact solutions. In particular, most NP-complete problems can be solved significantly faster than the (generic for the NP class) obvious brute-force algorithm that checks all possible solutions; examples are INDEPENDENT SET [11], DOMINATING SET [11, 23], CHROMATIC NUMBER [4] and BANDWIDTH [8]. The area of moderately exponential time algorithms studies upper and lower bounds for exact solutions for hard problems. The race for the fastest exact algorithm inspired several very interesting tools and techniques such as Fast Subset Convolution [3] and Measure&Conquer [11] (for an overview of the field we refer the reader to a recent book by Fomin and Kratsch [10]).

For several problems, including TSP, CHROMATIC NUMBER, PERMANENT, SET COVER, #HAMILTONIAN CYCLES and SAT, the currently best known time complexity is of the form[1] $O^*(2^n)$, which is a result of applying dynamic programming over subsets, the inclusion-exclusion principle or a brute force search. The question remains, however, which of those problems are inherently so hard that it is not possible to break the 2^n barrier and which are just waiting for new tools and techniques still to be discovered. In particular, the hardness of the k-SAT problem is the starting point for the Strong Exponential Time Hypothesis of Impagliazzo and Paturi [15], which is used as an argument that other problems are hard [7, 19, 22]. Recently, on the positive side, $O(c^n)$ time algorithms for a constant $c < 2$ have been developed for CAPACITATED DOMINATION [9], IRREDUNDANCE [1], MAXIMUM INDUCED PLANAR SUBGRAPH [12] and (a major breakthrough in the field) for the undirected version of the HAMILTONIAN CYCLE problem [2].

In this paper we extend this list by one important scheduling problem. The area of scheduling algorithms originates from practical questions regarding scheduling jobs on single- or multiple-processor machines or scheduling I/O requests. It has quickly become one of the most important areas in algorithmics, with significant influence on other branches of computer science. For example, the research of the job-shop scheduling problem in 1960s resulted in designing the competitive analysis [13], initiating the research of online algorithms. Up to today, the scheduling literature consists of thousands of research publications. We refer the reader to the classical textbook of Brucker [5].

Among scheduling problems one may find a bunch of problems solvable in polynomial time, as well as many NP-hard ones. For example, the aforementioned job-shop problem is NP-complete on at least three machines [17], but polynomial on two machines with unitary processing times [14].

Scheduling problems come in numerous variants. For example, one may consider scheduling on one machine, or many uniform or non-uniform machines. The jobs can have different attributes: they may arrive at different times, may have deadlines or precedence constraints, preemption may or may not be allowed. There are also many

---

[1] The $O^*()$ notation suppresses factors polynomial in the input size.

objective functions, for example the makespan of the computation, total completion time, total lateness (in case of deadlines for jobs) etc.

Let us focus on the case of a single machine. Assume we are given a set of jobs V, and each job v has its processing time $t(v) \in [0, +\infty)$. For a job v, its completion time is the total amount of time that this job waited to be finished; formally, the completion time of a job v is defined as the sum of processing times of v and all jobs scheduled earlier. If we are to minimize the total completion time (i.e, the sum of completion times over all jobs), it is clear that the jobs should be scheduled in order of increasing processing times. The question of minimizing the makespan of the computation (i.e., maximum completion time) is obvious in this setting, but we note that minimizing makespan is polynomially solvable even if we are given a precedence constraints on the jobs (i.e., a partial order on the set of jobs is given, and a job cannot be scheduled before all its predecessors in the partial order are finished) and jobs arrive at different times (i.e., each job has its arrival time, before which it cannot be scheduled) [16].

Lenstra and Rinnooy Kan [18] in 1978 proved that the question of minimizing total completion time on one machine becomes NP-complete if we are given precedence constraints on the set of jobs. To the best of our knowledge the currently smallest approximation ratio for this case equals 2, due to independently discovered algorithms by Chekuri and Motwani [6] as well as Margot et al. [20]. The problem of minimizing total completion time on one machine, given precedence constraints on the set of jobs, can be solved by a standard dynamic programming algorithm in time $O^*(2^n)$, where n denotes the number of jobs. In this paper we break the 2^n-barrier for this problem.

Before we start, let us define formally the considered problem. As we focus on a single scheduling problem, for brevity we denote it by SCHED. We note that the proper name of this problem in the Graham notation is $1|\mathrm{prec}|\sum C_i$.

SCHED

Input: A partially ordered set of jobs $(V, \leq)$, together with a nonnegative processing time $t(v) \in [0, +\infty)$ for each job $v \in V$.

Task: Compute a bijection $\sigma : V \to \{1, 2, \dots, |V|\}$ (called an *ordering*) that satisfies the precedence constraints (i.e., if $u < v$, then $\sigma(u) < \sigma(v)$) and minimizes the total completion time of all jobs defined as

$$T(\sigma) = \sum_{v \in V} \sum_{u:\sigma(u) \leq \sigma(v)} t(u) = \sum_{v \in V} \big(|V| - \sigma(v) + 1\big) t(v).$$

If $u < v$ for $u, v \in V$ (i.e., $u \leq v$ and $u \neq v$), we say that u *precedes* v, u is a *predecessor* or *prerequisite* of v, u is *required* for v or that v is a *successor* of u. We denote $|V|$ by n.

SCHED is a special case of the precedence constrained Traveling Repairman Problem (prec-TRP), defined as follows. A repairman needs to visit all vertices of a (directed or undirected) graph $G = (V, E)$ with distances $d : E \to [0, \infty)$ on edges. At each vertex, the repairman is supposed to repair a broken machine; a cost of a machine v is the time C_v that it waited before being repaired. Thus, the goal is to minimize the total repair time, that is, $\sum_{v \in V} C_v$. Additionally, in the precedence constrained case, we are given a partial order $(V, \leq)$ on the set of vertices of G; a machine can be repaired only if all its predecessors are already repaired. Note that, given an instance $(V, \leq, t)$ of SCHED, we may construct equivalent prec-TRP instance, by taking G to be a complete directed graph on the vertex set V, keeping the precedence constraints unmodified, and setting $d(u, v) = t(v)$.

The TRP problem is closely related to the Traveling Salesman Problem (TSP). All these problems are NP-complete and solvable in $O^*(2^n)$ time by an easy application of the dynamic programming approach (here n stands for the number of vertices in the input graph). In 2010, Björklund [2] discovered a genuine way to solve probably the easiest NP-complete version of the TSP problem—the question of deciding whether a given undirected graph is Hamiltonian—in randomized $O(1.66^n)$ time. However, his approach does not extend to directed graphs, not even mentioning graphs with distances defined on edges.

Björklund's approach is based on purely graph-theoretical and combinatorial reasonings, and seem unable to cope with arbitrary (large, real) weights (distances, processing times). This is also the case with many other combinatorial approaches. Probably motivated by this, Woeginger at International Workshop on Parameterized and Exact Computation (IWPEC) in 2004 [24] has posed the question (repeated in 2008 [25]), whether it is possible to construct an $O((2-\varepsilon)^n)$ time algorithm for the SCHED problem.[2] This problem seems to be the easiest case of the aforementioned family of TSP-related problems with arbitrary weights. In this paper we present such an algorithm, thus affirmatively answering Woeginger's question. Woeginger also asked [24, 25] whether an $O((2-\varepsilon)^n)$ time algorithm for one of the problems TRP, TSP, prec-TRP, SCHED implies $O((2-\varepsilon)^n)$ time algorithms for the other problems. This problem is still open.

The most important ingredient of our algorithm is a combinatorial lemma (Lemma 2.6) which allows us to investigate the structure of the SCHED problem. We heavily use the fact that we are solving the SCHED problem and not its more general TSP related version, and for this reason we believe that obtaining $O((2-\varepsilon)^n)$ time algorithms for other problems listed by Woeginger is much harder.

2 The Algorithm

2.1 High-Level Overview—Part 1

Let us recall that our task in the SCHED problem is to compute an ordering $\sigma : V \to \{1, 2, \ldots, n\}$ that satisfies the precedence constraints (i.e., if $u < v$ then $\sigma(u) < \sigma(v)$) and minimizes the total completion time of all jobs defined as

$$T(\sigma) = \sum_{v \in V} \sum_{u : \sigma(u) \leq \sigma(v)} t(u) = \sum_{v \in V} \bigl(n - \sigma(v) + 1\bigr) t(v).$$

---

[2] Although Woeginger in his papers asks for an $O(1.99^n)$ algorithm, the intention is clearly to ask for an $O((2-\varepsilon)^n)$ algorithm.

We define *the cost of job v at position i* to be $T(v,i)=(n-i+1)t(v)$. Thus, the total completion time is the total cost of all jobs at their respective positions in the ordering σ.

We begin by describing the algorithm that solves SCHED in $O^\star(2^n)$ time, which we call *the DP algorithm*—this will be the basis for our further work. The idea—a standard dynamic programming over subsets—is that if we decide that a particular set $X \subseteq V$ will (in some order) form the prefix of our optimal σ, then the order in which we take the elements of X does not affect the choices we make regarding the ordering of the remaining $V \setminus X$; the only thing that matters are the precedence constraints imposed by X on $V \setminus X$. Thus, for each candidate set $X \subseteq V$ to form a prefix, the algorithm computes a bijection $\sigma[X] : X \to \{1,2,\dots,|X|\}$ that minimizes the cost of jobs from X, i.e., it minimizes $T(\sigma[X]) = \sum_{v \in X} T(v, \sigma[X](v))$. The value of $T(\sigma[X])$ is computed using the following easy to check recursive formula:

$$T(\sigma[X]) = \min_{v \in \max(X)} \Big[T\big(\sigma\big[X \setminus \{v\}\big]\big) + T\big(v, |X|\big)\Big]. \tag{1}$$

Here, by $\max(X)$ we mean the set of maximum elements of X—those which do not precede any element of X. The bijection $\sigma[X]$ is constructed by prolonging $\sigma[X \setminus \{v\}]$ by v, where v is the job at which the minimum is attained. Notice that $\sigma[V]$ is exactly the ordering we are looking for. We calculate $\sigma[V]$ recursively, using formula (1), storing all computed values $\sigma[X]$ in memory to avoid recomputation. Thus, as the computation of a single $\sigma[X]$ value given all the smaller values takes polynomial time, while $\sigma[X]$ for each X is computed at most once the whole algorithm indeed runs in $O^\star(2^n)$ time.

The overall idea of our algorithm is to identify a family of sets $X \subseteq V$ that—for some reason—are not reasonable prefix candidates, and we can skip them in the computations of the DP algorithm; we will call these *unfeasible sets*. If the number of feasible sets is not larger than c^n for some $c < 2$, we will be done—our recursion will visit only feasible sets, assuming $T(\sigma[X])$ to be ∞ for unfeasible X in formula (1), and the running time will be $O^\star(c^n)$. This is formalized in the following proposition.

Proposition 2.1 *Assume we are given a polynomial-time algorithm $\mathcal{R}$ that, given a set $X \subseteq V$, either accepts it or rejects it. Moreover, assume that the number of sets accepted by $\mathcal{R}$ is bounded by $O(c^n)$ for some constant c. Then one can find in time $O^\star(c^n)$ an optimal ordering of the jobs in V among those orderings σ where $\sigma^{-1}(\{1,2,\dots,i\})$ is accepted by $\mathcal{R}$ for all $1 \le i \le n$, whenever such ordering exists.*

Proof Consider the following recursive procedure to compute optimal $T(\sigma[X])$ for a given set $X \subseteq V$:

1. if X is rejected by $\mathcal{R}$, return $T(\sigma[X]) = \infty$;
2. if $X = \emptyset$, return $T(\sigma[X]) = 0$;
3. if $T(\sigma[X])$ has been already computed, return the stored value of $T(\sigma[X])$;
4. otherwise, compute $T(\sigma[X])$ using formula (1), calling recursively the procedure itself to obtain values $T(\sigma[X \setminus \{v\}])$ for $v \in \max(X)$, and store the computed value for further use.

Clearly, the above procedure, invoked on $X = V$, computes optimal $T(\sigma[V])$ among those orderings σ where $\sigma^{-1}(\{1, 2, \ldots, i\})$ is accepted by $\mathcal{R}$ for all $1 \leq i \leq n$. It is straightforward to augment this procedure to return the ordering σ itself, instead of only its cost.

If we use balanced search tree to store the computed values of $\sigma[X]$, each recursive call of the described procedure runs in polynomial time. Note that the last step of the procedure is invoked at most once for each set X accepted by $\mathcal{R}$ and never for a set X rejected by $\mathcal{R}$. As an application of this step results in at most $|X| \leq n$ recursive calls, we obtain that a computation of $\sigma[V]$ using this procedure results in the number of recursive calls bounded by n times the number of sets accepted by $\mathcal{R}$. The time bound follows. ☐

2.2 The Large Matching Case

We begin by noticing that the DP algorithm needs to compute $\sigma[X]$ only for those $X \subseteq V$ that are downward closed, i.e., if $v \in X$ and $u < v$ then $u \in X$. If there are many constraints in our problem, this alone will suffice to limit the number of feasible sets considerably, as follows. Construct an undirected graph G with the vertex set V and edge set $E = \{uv : u < v \vee v < u\}$. Let $\mathcal{M}$ be a maximum matching[3] in G, which can be found in polynomial time [21]. If $X \subseteq V$ is downward closed, and $uv \in \mathcal{M}$, $u < v$, then it is not possible that $u \notin X$ and $v \in X$. Obviously checking if a subset is downward closed can be performed in polynomial time, thus we can apply Proposition 2.1, accepting only downward closed subsets of V. This leads to the following lemma:

Lemma 2.2 *The number of downward closed subsets of* V *is bounded by* $2^{n-2|\mathcal{M}|} \times 3^{|\mathcal{M}|}$. *If* $|\mathcal{M}| \geq \varepsilon_1 n$, *then we can solve the* SCHED *problem in time*

$$T_1(n) = O^\star\big((3/4)^{\varepsilon_1 n} 2^n\big).$$

Note that for any small positive constant ε_1 the complexity $T_1(n)$ is of required order, i.e., $T_1(n) = O(c^n)$ for some $c < 2$ that depends on ε_1. Thus, we only have to deal with the case where $|\mathcal{M}| < \varepsilon_1 n$.

Let us fix a maximum matching $\mathcal{M}$, let $M \subseteq V$ be the set of endpoints of $\mathcal{M}$, and let $I_1 = V \setminus M$. Note that, as M is a maximum matching in G, no two jobs in I_1 are bound by a precedence constraint, and $|M| \leq 2\varepsilon_1 n$, $|I_1| \geq (1 - 2\varepsilon_1)n$. See Fig. 1 for an illustration.

2.3 High-Level Overview—Part 2

We are left in the situation where there is a small number of "special" elements (M), and the bulk remainder (I_1), consisting of elements that are tied by precedence constraints only to M and not to each other.

---

[3] Even an inclusion-maximal matching, which can be found greedily, is enough.

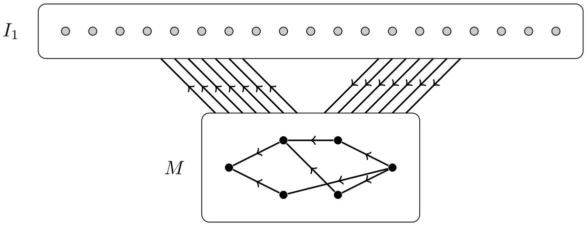


Fig. 1 An illustration of the case left after Lemma 2.2. In this and all further figures, an arrow points from the successor job to the predecessor one

First notice that if M was empty, the problem would be trivial: with no precedence constraints we should simply order the tasks from the shortest to the longest. Now let us consider what would happen if all the constraints between any $u \in I_1$ and $w \in M$ would be of the form $u < w$—that is, if the jobs from I_1 had no predecessors. For any prefix set candidate X we consider $X_I = X \cap I_1$. Now for any $x \in X_I$, $y \in I_1 \setminus X_I$ we have an alternative prefix candidate: the set $X' = (X \cup \{y\}) \setminus \{x\}$. If $t(y) < t(x)$, there has to be a reason why X' is not a strictly better prefix candidate than X—namely, there has to exist $w \in M$ such that $x < w$, but $y \not< w$.

A similar reasoning would hold even if not all of I_1 had no predecessors, but just some constant fraction J of I—again, the only feasible prefix candidates would be those in which for every $x \in X_I \cap J$ and $y \in J \setminus X_I$ there is a reason (either $t(x) < t(y)$ or an element $w \in M$ which requires x, but not y) not to exchange them. It turns out that if $|J| > \varepsilon_2 n$, where $\varepsilon_2 > 2\varepsilon_1$, this observation suffices to prove that the number of possible intersections of feasible sets with J is exponentially smaller than $2^{|J|}$. This is formalized and proved in Lemma 2.6, and is the cornerstone of the whole result.

A typical application of this lemma is as follows: say we have a set $K \subseteq I_1$ of cardinality $|K| > 2j$, while we know for some reason that all the predecessors of elements of K appear on positions j and earlier. If K is large (a constant fraction of n), this is enough to limit the number of feasible sets to $(2-\varepsilon)^n$. To this end it suffices to show that there are exponentially fewer than $2^{|K|}$ possible intersections of a feasible set with K. Each such intersection consists of a set of at most j elements (that will be put on positions 1 through j), and then a set in which every element has a reason not to be exchanged with something from outside the set—and there are relatively few of those by Lemma 2.6—and when we do the calculations, it turns out the resulting number of possibilities is exponentially smaller than $2^{|K|}$.

To apply this reasoning, we need to be able to tell that all the prerequisites of a given element appear at some position or earlier. To achieve this, we need to know the approximate positions of the elements in M. We achieve this by branching into $4^{|M|}$ cases, for each element $w \in M$ choosing to which of the four quarters of the set $\{1, \dots, n\}$ will $\sigma_{opt}(w)$ belong. This incurs a multiplicative cost[4] of $4^{|M|}$, which will be offset by the gains from applying Lemma 2.6.

We will now repeatedly apply Lemma 2.6 to obtain information about the positions of various elements of I_1. We will repeatedly say that if "many" elements (by which

---

[4] Actually, this bound can be improved to $10^{|M|/2}$, as M are endpoints of a matching in the graph corresponding to the set of precedences.

we always mean more than εn for some ε) do not satisfy something, we can bound the number of feasible sets, and thus finish the algorithm. For instance, look at those elements of I_1 which can appear in the first quarter, i.e., none of their prerequisites appear in quarters two, three and four. If there is more than $(\frac{1}{2}+\delta)n$ of them for some constant $\delta > 0$, we can apply the above reasoning for $j = n/4$ (Lemma 2.10). Subsequent lemmata bound the number of feasible sets if there are many elements that cannot appear in any of the two first quarters (Lemma 2.8), if *less* than $(\frac{1}{2}-\delta)n$ elements can appear in the first quarter (Lemma 2.10) and if a constant fraction of elements in the second quarter could actually appear in the first quarter (Lemma 2.11). We also apply similar reasoning to elements that can or cannot appear in the last quarter.

We end up in a situation where we have four groups of elements, each of size roughly $n/4$, split upon whether they can appear in the first quarter and whether they can appear in the last one; moreover, those that can appear in the first quarter will not appear in the second, and those that can appear in the fourth will not appear in the third. This means that there are two pairs of parts which do not interact, as the set of places in which they can appear are disjoint. We use this independence of sorts to construct a different algorithm than the DP we used so far, which solves our problem in this specific case in time $O^\star(2^{3n/4+\varepsilon})$ (Lemma 2.12).

As can be gathered from this overview, there are many technical details we will have to navigate in the algorithm. This is made more precarious by the need to carefully select all the epsilons. We decided to use symbolic values for them in the main proof, describing their relationship appropriately, using four constants ε_k, $k = 1, 2, 3, 4$. The constants ε_k are very small positive reals, and additionally ε_k is much smaller than ε_{k+1} for $k = 1, 2, 3$. At each step, we shortly discuss the existence of such constants. We discuss the choice of optimal values of these constants in Sect. 2.9, although the value we perceive in our algorithm lies rather in the existence of an $O^\star((2-\varepsilon)^n)$ algorithm than in the value of ε (which is admittedly very small).

2.4 Technical Preliminaries

We start with a few simplifications. First, we add a few dummy jobs with no precedence constraints and zero processing times, so that n is divisible by four. Second, by slightly perturbing the jobs' processing times, we can assume that all processing times are pairwise different and, moreover, each ordering has different total completion time. This can be done, for instance, by replacing time $t(v)$ with a pair $(t(v), (n+1)^{\pi(v)-1})$, where $\pi : V \to \{1, 2, \dots, n\}$ is an arbitrary numbering of V. The addition of pairs is performed coordinatewise, whereas comparison is performed lexicographically. Note that this in particular implies that the optimal solution is unique, we denote it by σ_{opt}. Third, at the cost of an n^2 multiplicative overhead, we guess the jobs $v_{begin} = \sigma_{opt}^{-1}(1)$ and $v_{end} = \sigma_{opt}^{-1}(n)$ and we add precedence constraints $v_{begin} < v < v_{end}$ for each $v \neq v_{begin}, v_{end}$. If v_{begin} or v_{end} were not in M to begin with, we add them there.

A number of times our algorithm branches into several subcases, in each branch assuming some property of the optimal solution σ_{opt}. Formally speaking, in each branch we seek the optimal ordering among those that satisfy the assumed property.

We somewhat abuse the notation and denote by σ_{opt} the optimal solution in the currently considered subcase. Note that σ_{opt} is always unique within any subcase, as each ordering has different total completion time.

For $v \in V$ by $pred(v)$ we denote the set $\{u \in V : u < v\}$ of predecessors of v, and by $succ(v)$ we denote the set $\{u \in V : v < u\}$ of successors of v. We extend this notation to subsets of V: $pred(U) = \bigcup_{v \in U} pred(v)$ and $succ(U) = \bigcup_{v \in U} succ(v)$. Note that for any set $U \subseteq I_1$, both $pred(U)$ and $succ(U)$ are subsets of M.

In a few places in this paper we use the following simple bound on binomial coefficients that can be easily proven using the Stirling's formula.

Lemma 2.3 *Let* $0 < \alpha < 1$ *be a constant. Then*

$$\binom{n}{\alpha n} = O^*\left(\left(\frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}}\right)^n\right).$$

In particular, if $\alpha \neq 1/2$ *then there exists a constant* $c_\alpha < 2$ *that depends only on* α *and*

$$\binom{n}{\alpha n} = O^*\left(c_\alpha^n\right).$$

2.5 The Core Lemma

We now formalize the idea of exchanges presented at the beginning of Sect. 2.3.

Definition 2.4 Consider some set $K \subseteq I_1$, and its subset $L \subseteq K$. If there exists $u \in L$ such that for every $w \in succ(u)$ we can find $v_w \in (K \cap pred(w)) \setminus L$ with $t(v_w) < t(u)$ then we say L is *succ-exchangeable* with respect to K, otherwise we say L is *non-succ-exchangeable* with respect to K.

Similarly, if there exists $v \in (K \setminus L)$ such that for every $w \in pred(v)$ we can find $u_w \in L \cap succ(w)$ with $t(u_w) > t(v)$, we call L *pred-exchangeable* with respect to K, otherwise we call it *non-pred-exchangeable* with respect to K.

Whenever it is clear from the context, we omit the set K with respect to which its subset is or is not *pred*- or *succ*-exchangeable.

Let us now give some more intuition on the exchangeable sets. Let L be a non-*succ*-exchangeable set with respect to $K \subseteq I_1$ and let $u \in L$. By the definition, there exists $w \in succ(u)$, such that for all $v_w \in (K \cap pred(w)) \setminus L$ we have $t(v_w) \geq t(u)$; in other words, all predecessors of w in K that are scheduled after L have larger processing time than u—which seems like a "correct" choice if we are to optimize the total completion time.

On the other hand, let $L = \sigma_{opt}^{-1}(\{1, 2, \ldots, i\}) \cap K$ for some $1 \leq i \leq n$ and assume that L is a *succ*-exchangeable set with respect to K with a job $u \in L$ witnessing this fact. Let w be the job in $succ(u)$ that is scheduled first in the optimal ordering σ_{opt}. By the definition, there exists $v_w \in (K \cap pred(w)) \setminus L$ with $t(v_w) < t(u)$. It is tempting to decrease the total completion time of σ_{opt} by swapping the jobs v_w and u in σ_{opt}: by the choice of w, no precedence constraint involving u will be violated by such an exchange, so we need to care only about the predecessors of v_w.

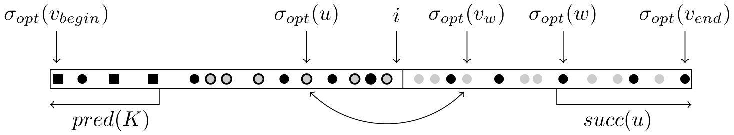


Fig. 2 Figure illustrating the *succ*-exchangeable case of Lemma 2.5. *Gray circles* indicate positions of elements of K, *black contour* indicates that an element is also in L. *Black squares* indicate positions of elements from $pred(K)$, and *black circles*—positions of other elements from M

We formalize the aforementioned applicability of the definition of *pred*- and *succ*-exchangeable sets in the following lemma:

Lemma 2.5 *Let $K \subseteq I_1$. If for all $v \in K, x \in pred(K)$ we have that $\sigma_{opt}(v) > \sigma_{opt}(x)$, then for any $1 \le i \le n$ the set $K \cap \sigma_{opt}^{-1}(\{1, 2, \dots, i\})$ is non-succ-exchangeable with respect to K.*

Similarly, if for all $v \in K, x \in succ(K)$ we have $\sigma_{opt}(v) < \sigma_{opt}(x)$, then the sets $K \cap \sigma_{opt}^{-1}(\{1, 2, \dots, i\})$ are non-pred-exchangeable with respect to K.

Proof The proofs for the first and the second case are analogous. However, to help the reader get intuition on exchangeable sets, we provide them both in full detail. See Fig. 2 for an illustration on the *succ*-exchangeable case.

Non-succ-exchangeable sets. Assume, by contradiction, that for some i the set $L = K \cap \sigma_{opt}^{-1}(\{1, 2, \dots, i\})$ is *succ*-exchangeable. Let $u \in L$ be a job witnessing it. Let w be the successor of u with minimum $\sigma_{opt}(w)$ (there exists one, as $v_{end} \in succ(u)$). By Definition 2.4, we have $v_w \in (K \cap pred(w)) \setminus L$ with $t(v_w) < t(u)$. As $v_w \in K \setminus L$, we have $\sigma_{opt}(v_w) > \sigma_{opt}(u)$. As $v_w \in pred(w)$, we have $\sigma_{opt}(v_w) < \sigma_{opt}(w)$.

Consider an ordering σ' defined as $\sigma'(u) = \sigma_{opt}(v_w)$, $\sigma'(v_w) = \sigma_{opt}(u)$ and $\sigma'(x) = \sigma_{opt}(x)$ if $x \notin \{u, v_w\}$; in other words, we swap the positions of u and v_w in the ordering σ_{opt}. We claim that σ' satisfies all the precedence constraints. As $\sigma_{opt}(u) < \sigma_{opt}(v_w)$, σ' may only violates constraints of the form $x < v_w$ and $u < y$. However, if $x < v_w$, then $x \in pred(K)$ and $\sigma'(v_w) = \sigma_{opt}(u) > \sigma_{opt}(x) = \sigma'(x)$ by the assumptions of the Lemma. If $u < y$, then $\sigma'(y) = \sigma_{opt}(y) \ge \sigma_{opt}(w) > \sigma_{opt}(v_w) = \sigma'(u)$, by the choice of w. Thus σ' is a feasible solution to the considered SCHED instance. Since $t(v_w) < t(u)$, we have $T(\sigma') < T(\sigma_{opt})$, a contradiction.

Non-pred-exchangeable sets. Assume, by contradiction, that for some i the set $L = K \cap \sigma_{opt}^{-1}(\{1, 2, \dots, i\})$ is *pred*-exchangeable. Let $v \in (K \setminus L)$ be a job witnessing it. Let w be the predecessor of v with maximum $\sigma_{opt}(w)$ (there exists one, as $v_{begin} \in pred(v)$). By Definition 2.4, we have $u_w \in L \cap succ(w)$ with $t(u_w) > t(v)$. As $u_w \in L$, we have $\sigma_{opt}(u_w) < \sigma_{opt}(v)$. As $u_w \in succ(w)$, we have $\sigma_{opt}(u_w) > \sigma_{opt}(w)$.

Consider an ordering σ' defined as $\sigma'(v) = \sigma_{opt}(u_w)$, $\sigma'(u_w) = \sigma_{opt}(v)$ and $\sigma'(x) = \sigma_{opt}(x)$ if $x \notin \{v, u_w\}$; in other words, we swap the positions of v and u_w in the ordering σ_{opt}. We claim that σ' satisfies all the precedence constraints. As $\sigma_{opt}(u_w) < \sigma_{opt}(v)$, σ' may only violates constraints of the form $x > u_w$ and $v > y$. However, if $x > u_w$, then $x \in succ(K)$ and $\sigma'(u_w) = \sigma_{opt}(v) < \sigma_{opt}(x) = \sigma'(x)$

by the assumptions of the Lemma. If $v > y$, then $\sigma'(y) = \sigma_{opt}(y) \leq \sigma_{opt}(w) < \sigma_{opt}(u_w) = \sigma'(v)$, by the choice of w. Thus σ' is a feasible solution to the considered SCHED instance. Since $t(u_w) > t(v)$, we have $T(\sigma') < T(\sigma_{opt})$, a contradiction. □

Lemma 2.5 means that if we manage to identify a set K satisfying the assumptions of the lemma, the only sets the DP algorithm has to consider are the non-exchangeable ones. The following core lemma proves that there are few of those (provided that K is big enough), and we can identify them easily.

Lemma 2.6 *For any set $K \subseteq I_1$ the number of non-succ-exchangeable (non-pred-exchangeable) subsets with regard to K is at most $\sum_{l \leq |M|} \binom{|K|}{l}$. Moreover, there exists an algorithm which checks whether a set is succ-exchangeable (pred-exchangeable) in polynomial time.*

The idea of the proof is to construct a function f that encodes each non-exchangeable set by a subset of K no larger than M. To show this encoding is injective, we provide a decoding function g and show that $g \circ f$ is an identity on non-exchangeable sets.

Proof As in Lemma 2.5, the proofs for *succ*- and *pred*-exchangeable sets are analogous, but for the sake or clarity we include both proofs in full detail.

Non-succ-exchangeable sets. For any set $Y \subseteq K$ we define the function $f_Y : M \to K \cup \{\text{nil}\}$ as follows: for any element $w \in M$ we define $f_Y(w)$ (the *least expensive predecessor of w outside Y*) to be the element of $(K \setminus Y) \cap pred(w)$ which has the smallest processing time, or nil if $(K \setminus Y) \cap pred(w)$ is empty. We now take $f(Y)$ (the set of the *least expensive predecessors outside Y*) to be the set $\{f_Y(w) : w \in M\} \setminus \{\text{nil}\}$. We see that $f(Y)$ is indeed a set of cardinality at most $|M|$.

Now we aim to prove that f is injective on the family of non-*succ*-exchangeable sets. To this end we define the reverse function g. For a set $Z \subseteq K$ (which we think of as the set of the least expensive predecessors outside some Y) let $g(Z)$ be the set of such elements v of K that there exists $w \in succ(v)$ such that for any $z_w \in Z \cap pred(w)$ we have $t(z_w) > t(v)$. Notice, in particular, that $g(Z) \cap Z = \emptyset$, as for $v \in Z$ and $w \in succ(v)$ we have $v \in Z \cap pred(w)$.

First we prove $g(f(Y)) \subseteq Y$ for any $Y \subseteq K$. Take any $v \in K \setminus Y$ and consider any $w \in succ(v)$. Then $f_Y(w) \neq \text{nil}$ and $t(f_Y(w)) \leq t(v)$, as $v \in (K \setminus Y) \cap pred(w)$. Thus $v \notin g(f(Y))$, as for any $w \in succ(v)$ we can take a witness $z_w = f_Y(w)$ in the definition of $g(f(Y))$.

In the other direction, let us assume that Y does not satisfy $Y \subseteq g(f(Y))$. This means we have $u \in Y \setminus g(f(Y))$. Then we show that Y is *succ*-exchangeable. Consider any $w \in succ(u)$. As $u \notin g(f(Y))$, by the definition of the function g applied to the set $f(Y)$, there exists $z_w \in f(Y) \cap pred(w)$ with $t(z_w) \leq t(u)$. But $f(Y) \cap Y = \emptyset$, while $u \in Y$; and as all the values of t are distinct, $t(z_w) < t(u)$ and z_w satisfies the condition for v_w in the definition of *succ*-exchangeability.

Non-pred-exchangeable sets. For any set $Y \subseteq K$ we define the function $f_Y : M \to K \cup \{\text{nil}\}$ as follows: for any element $w \in M$ we define $f_Y(w)$ (the *most expensive successor of w in Y*) to be the element of $Y \cap succ(w)$ which has the largest processing time, or nil if $Y \cap succ(w)$ is empty. We now take $f(Y)$ (the set of the *most* *expensive successors in* Y) to be the set $\{f_Y(w) : w \in M\} \setminus \{\text{nil}\}$. We see that $f(Y)$ is indeed a set of cardinality at most $|M|$.

Now we aim to prove that f is injective on the family of non-*pred*-exchangeable sets. To this end we define the reverse function g. For a set $Z \subseteq K$ (which we think of as the set of most expensive successors in some Y) let $g(Z)$ be the set of such elements v of K that for any $w \in \mathit{pred}(v)$ there exists a $z_w \in Z \cap \mathit{succ}(w)$ with $t(z_w) \geq t(v)$. Notice, in particular, that $g(Z) \subseteq Z$, as for $v \in Z$ the job $z_w = v$ is a good witness for any $w \in \mathit{pred}(v)$.

First we prove $Y \subseteq g(f(Y))$ for any $Y \subseteq K$. Take any $v \in Y$ and consider any $w \in \mathit{pred}(v)$. Then $f_Y(w) \neq \text{nil}$ and $t(f_Y(w)) \geq t(v)$, as $v \in Y \cap \mathit{succ}(w)$. Thus $v \in g(f(Y))$, as for any $w \in \mathit{pred}(v)$ we can take $z_w = f_Y(w)$ in the definition of $g(f(Y))$.

In the other direction, let us assume that Y does not satisfy $g(f(Y)) \subseteq Y$. This means we have $v \in g(f(Y)) \setminus Y$. Then we show that Y is *pred*-exchangeable. Consider any $w \in \mathit{pred}(v)$. As $v \in g(f(Y))$, by the definition of the function g applied to the set $f(Y)$, there exists $z_w \in f(Y) \cap \mathit{succ}(w)$ with $t(z_w) \geq t(v)$. But $f(Y) \subseteq Y$, while $v \notin Y$; and as all the values of t are distinct, $t(z_w) > t(v)$ and z_w satisfies the condition for u_w in the definition of *pred*-exchangeability.

Thus, in both cases, if Y is non-exchangeable then $g(f(Y)) = Y$ (in fact it is possible to prove in both cases that Y is non-exchangeable iff $g(f(Y)) = Y$). As there are $\sum_{l=0}^{|M|} \binom{|K|}{l}$ possible values of $f(Y)$, the first part of the lemma is proven. For the second, it suffices to notice that *succ*- and *pred*-exchangeability can be checked in time $O(|K|^2|M|)$ directly from the definition. ☐

Example 2.7 To illustrate the applicability of Lemma 2.6, we analyze the following very simple case: assume the whole set $M \setminus \{v_{begin}\}$ succeeds I_1, i.e., for every $w \in M \setminus \{v_{begin}\}$ and $v \in I_1$ we have $w \not< v$. If ε_1 is small, then we can use the first case of Lemma 2.5 for the whole set $K = I_1$: we have $\mathit{pred}(K) = \{v_{begin}\}$ and we only look for orderings that put v_{begin} as the first processed job. Thus, we can apply Proposition 2.1 with algorithm $\mathcal{R}$ that rejects sets $X \subseteq V$ where $X \cap I_1$ is *succ*-exchangeable with respect to I_1. By Lemma 2.6, the number of sets accepted by $\mathcal{R}$ is bounded by $2^{|M|} \sum_{l \leq |M|} \binom{|I_1|}{l}$, which is small if $|M| \leq \varepsilon_1 n$.

2.6 Important Jobs at $n/2$

As was already mentioned in the overview, the assumptions of Lemma 2.5 are quite strict; therefore, we need to learn a bit more on how σ_{opt} behaves on M in order to distinguish a suitable place for an application. As $|M| \leq 2\varepsilon_1 n$, we can afford branching into few subcases for every job in M.

Let $A = \{1, 2, \ldots, n/4\}$, $B = \{n/4 + 1, \ldots, n/2\}$, $C = \{n/2 + 1, \ldots, 3n/4\}$, $D = \{3n/4 + 1, \ldots, n\}$, i.e., we split $\{1, 2, \ldots, n\}$ into quarters. For each $w \in M \setminus \{v_{begin}, v_{end}\}$ we branch into two cases: whether $\sigma_{opt}(w)$ belongs to $A \cup B$ or $C \cup D$; however, if some predecessor (successor) of w has been already assigned to $C \cup D$ ($A \cup B$), we do not allow w to be placed in $A \cup B$ ($C \cup D$). Of course, we already know that $\sigma_{opt}(v_{begin}) \in A$ and $\sigma_{opt}(v_{end}) \in D$. Recall that the vertices of M can be paired into a matching; since for each $w_1 < w_2$, $w_1, w_2 \in M$ we cannot have

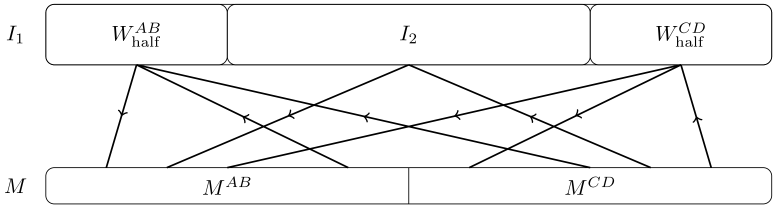


Fig. 3 An illustration of the sets M^{AB}, M^{CD}, W^{AB}_{half} and W^{CD}_{half}

w_1 placed in $C \cup D$ and w_2 placed in $A \cup B$, this branching leads to $3^{|M|/2} \le 3^{\varepsilon_1 n}$ subcases, and thus the same overhead in the time complexity. By the above procedure, in all branches the guesses about alignment of jobs from M satisfy precedence constraints inside M.

Now consider a fixed branch. Let M^{AB} and M^{CD} be the sets of elements of M to be placed in $A \cup B$ and $C \cup D$, respectively.

Let us now see what we can learn in a fixed branch about the behavior of σ_{opt} on I_1. Let

$$W^{AB}_{\text{half}} = \bigl\{v \in I_1 : \exists_w \bigl(w \in M^{AB} \wedge v < w\bigr)\bigr\},$$
$$W^{CD}_{\text{half}} = \bigl\{v \in I_1 : \exists_w \bigl(w \in M^{CD} \wedge w < v\bigr)\bigr\},$$

that is W^{AB}_{half} (resp. W^{CD}_{half}) are those elements of I_1 which are forced into the first (resp. second) half of σ_{opt} by the choices we made about M (see Fig. 3 for an illustration). If one of the W_{half} sets is much larger than M, we have obtained a gain—by branching into at most $3^{\varepsilon_1 n}$ branches we gained additional information about a significant (much larger than $(\log_2 3)\varepsilon_1 n$) number of other elements (and so we will be able to avoid considering a significant number of sets in the DP algorithm). This is formalized in the following lemma:

Lemma 2.8 *Consider a fixed branch. If W^{AB}_{half} or W^{CD}_{half} has at least $\varepsilon_2 n$ elements, then the DP algorithm can be augmented to solve the instance in the considered branch in time*

$$T_2(n) = \left(2^{(1-\varepsilon_2)n} + \binom{n}{(1/2-\varepsilon_2)n} + 2^{\varepsilon_2 n}\binom{(1-\varepsilon_2)n}{n/2}\right)n^{O(1)}.$$

Proof We describe here only the case $|W^{AB}_{\text{half}}| \ge \varepsilon_2 n$. The second case is symmetrical.

Recall that the set W^{AB}_{half} needs to be placed in $A \cup B$ by the optimal ordering σ_{opt}. We use Proposition 2.1 with an algorithm $\mathcal{R}$ that accepts sets $X \subseteq V$ such that the set $W^{AB}_{\text{half}} \setminus X$ (the elements of W^{AB}_{half} not scheduled in X) is of size at most $\max(0, n/2 - |X|)$ (the number of jobs to be scheduled after X in the first half of the jobs). Moreover, the algorithm $\mathcal{R}$ tests if the set X conforms with the guessed sets M^{AB} and M^{CD}, i.e.:

$$|X| \le n/2 \quad \Rightarrow \quad M^{CD} \cap X = \emptyset,$$

$$|X| \geq n/2 \quad \Rightarrow \quad M^{AB} \subseteq X.$$

Clearly, for any $1 \leq i \leq n$, the set $\sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ is accepted by $\mathcal{R}$, as σ_{opt} places $M^{AB} \cup W_{\mathrm{half}}^{AB}$ in $A \cup B$ and M^{CD} in $C \cup D$.

Let us now estimate the number of sets X accepted by $\mathcal{R}$. Any set X of size larger than $n/2$ needs to contain W_{half}^{AB}; there are at most $2^{n-|W_{\mathrm{half}}^{AB}|} \leq 2^{(1-\varepsilon_2)n}$ such sets. All sets of size at most $n/2 - |W_{\mathrm{half}}^{AB}|$ are accepted by $\mathcal{R}$; there are at most $n\binom{n}{(1/2-\varepsilon_2)n}$ such sets. Consider now a set X of size $n/2-\alpha$ for some $0 \leq \alpha \leq |W_{\mathrm{half}}^{AB}|$. Such a set needs to contain $|W_{\mathrm{half}}^{AB}| - \beta$ elements of W_{half}^{AB} for some $0 \leq \beta \leq \alpha$ and $n/2 - |W_{\mathrm{half}}^{AB}| - (\alpha - \beta)$ elements of $V \setminus W_{\mathrm{half}}^{AB}$. Therefore the number of such sets (for all possible α) is bounded by:

$$\begin{aligned}
&\sum_{\alpha=0}^{|W_{\mathrm{half}}^{AB}|} \sum_{\beta=0}^{\alpha} \binom{|W_{\mathrm{half}}^{AB}|}{|W_{\mathrm{half}}^{AB}| - \beta} \binom{n - |W_{\mathrm{half}}^{AB}|}{n/2 - |W_{\mathrm{half}}^{AB}| - (\alpha - \beta)} \\
&\quad \leq n^2 \max_{0 \leq \beta \leq \alpha \leq |W_{\mathrm{half}}^{AB}|} \binom{|W_{\mathrm{half}}^{AB}|}{\beta} \binom{n - |W_{\mathrm{half}}^{AB}|}{n/2 + (\alpha - \beta)} \\
&\quad \leq n^2 2^{|W_{\mathrm{half}}^{AB}|} \binom{n - |W_{\mathrm{half}}^{AB}|}{n/2} \\
&\quad \leq n^2 2^{\varepsilon_2 n} \binom{(1-\varepsilon_2)n}{n/2}.
\end{aligned}$$

The last inequality follows from the fact that the function $x \mapsto 2^x \binom{n-x}{n/2}$ is decreasing for $x \in [0, n/2]$. The bound $T_2(n)$ follows.

□

Note that we have $3^{\varepsilon_1 n}$ overhead so far, due to guessing placement of the jobs from M. By Lemma 2.3, $\binom{(1-\varepsilon_2)n}{n/2} = O((2 - c(\varepsilon_2))^{(1-\varepsilon_2)n})$ and $\binom{n}{(1/2-\varepsilon_2)n} = O((2 - c'(\varepsilon_2))^n)$, for some positive constants $c(\varepsilon_2)$ and $c'(\varepsilon_2)$ that depend only on ε_2. Thus, for any small fixed ε_2 we can choose ε_1 sufficiently small so that $3^{\varepsilon_1 n} T_2(n) = O(c^n)$ for some $c < 2$. Note that $3^{\varepsilon_1 n} T_2(n)$ is an upper bound on the total time spent on processing all the considered subcases.

Let $W_{\mathrm{half}} = W_{\mathrm{half}}^{AB} \cup W_{\mathrm{half}}^{CD}$ and $I_2 = I_1 \setminus W_{\mathrm{half}}$. From this point we assume that $|W_{\mathrm{half}}^{AB}|, |W_{\mathrm{half}}^{CD}| \leq \varepsilon_2 n$, hence $|W_{\mathrm{half}}| \leq 2\varepsilon_2 n$ and $|I_2| \geq (1 - 2\varepsilon_1 - 2\varepsilon_2)n$. For each $v \in M^{AB} \cup W_{\mathrm{half}}^{AB}$ we branch into two subcases, whether $\sigma_{opt}(v)$ belongs to A or B. Similarly, for each $v \in M^{CD} \cup W_{\mathrm{half}}^{CD}$ we guess whether $\sigma_{opt}(v)$ belongs to C or D. Moreover, we terminate branches which are trivially contradicting the constraints.

Let us now estimate the number of subcases created by this branch. Recall that the vertices of M can be paired into a matching; since for each $w_1 < w_2$, $w_1, w_2 \in M$ we cannot have w_1 placed in a later segment than w_2; this gives us 10 options for each pair $w_1 < w_2$. Thus, in total they are at most $10^{|M|/2} \leq 10^{\varepsilon_1 n}$ ways of placing vertices of M into quarters without contradicting the constraints. Moreover, this step gives us an additional $2^{|W_{\text{half}}|} \le 2^{2\varepsilon_2 n}$ overhead in the time complexity for vertices in W_{half}. Overall, at this point we are considering at most $10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} n^{O(1)}$ subcases.

We denote the set of elements of M and W_{half} assigned to quarter $\Gamma \in \{A, B, C, D\}$ by M^Γ and W^Γ_{half}, respectively.

2.7 Quarters and Applications of the Core Lemma

In this section we try to apply Lemma 2.6 as follows: We look which elements of I_2 can be placed in A (the set P^A) and which cannot (the set $P^{\neg A}$). Similarly we define the set P^D (can be placed in D) and $P^{\neg D}$ (cannot be placed in D). For each of these sets, we try to apply Lemma 2.6 to some subset of it. If we fail, then in the next subsection we infer that the solutions in the quarters are partially independent of each other, and we can solve the problem in time roughly $O(2^{3n/4})$. Let us now proceed with a more detailed argumentation.

We define the following two partitions of I_2:

$$\begin{aligned}
P^{\neg A} &= \bigl\{v \in I_2 : \exists_w \bigl(w \in M^B \wedge w < v\bigr)\bigr\}, \\
P^A &= I_2 \setminus P^{\neg A} = \bigl\{v \in I_2 : \forall_w \bigl(w < v \Rightarrow w \in M^A\bigr)\bigr\}, \\
P^{\neg D} &= \bigl\{v \in I_2 : \exists_w \bigl(w \in M^C \wedge w > v\bigr)\bigr\}, \\
P^D &= I_2 \setminus P^{\neg D} = \bigl\{v \in I_2 : \forall_w \bigl(w > v \Rightarrow w \in M^D\bigr)\bigr\}.
\end{aligned}$$

In other words, the elements of $P^{\neg A}$ cannot be placed in A because some of their requirements are in M^B, and the elements of $P^{\neg D}$ cannot be placed in D because they are required by some elements of M^C (see Fig. 4 for an illustration). Note that these definitions are independent of σ_{opt}, so sets P^Δ for $\Delta \in \{A, \neg A, \neg D, D\}$ can be computed in polynomial time. Let

$$\begin{aligned}
p^A &= \bigl|\sigma_{opt}\bigl(P^A\bigr) \cap A\bigr|, \\
p^B &= \bigl|\sigma_{opt}\bigl(P^{\neg A}\bigr) \cap B\bigr|, \\
p^C &= \bigl|\sigma_{opt}\bigl(P^{\neg D}\bigr) \cap C\bigr|, \\
p^D &= \bigl|\sigma_{opt}\bigl(P^D\bigr) \cap D\bigr|.
\end{aligned}$$

Note that $p^\Gamma \le n/4$ for every $\Gamma \in \{A, B, C, D\}$. As $p^A = n/4 - |M^A \cup W^A_{\text{half}}|$, $p^D = n/4 - |M^D \cup W^D_{\text{half}}|$, these values can be computed by the algorithm. We branch into $(1 + n/4)^2$ further subcases, guessing the (still unknown) values p^B and p^C.

Let us focus on the quarter A and assume that p^A is significantly smaller than $|P^A|/2$ (i.e., $|P^A|/2 - p^a$ is a constant fraction of n). We claim that we can apply Lemma 2.6 as follows. While computing $\sigma[X]$, if $|X| \ge n/4$, we can represent $X \cap P^A$ as a disjoint sum of two subsets $X^A_A, X^A_{BCD} \subseteq P^A$. The first one is of size p^A, and represents the elements of $X \cap P^A$ placed in quarter A, and the second represents the elements of $X \cap P^A$ placed in quarters $B \cup C \cup D$. Note that the elements of X^A_{BCD} have all predecessors in the quarter A, so by Lemma 2.5 the set X^A_{BCD} has to be

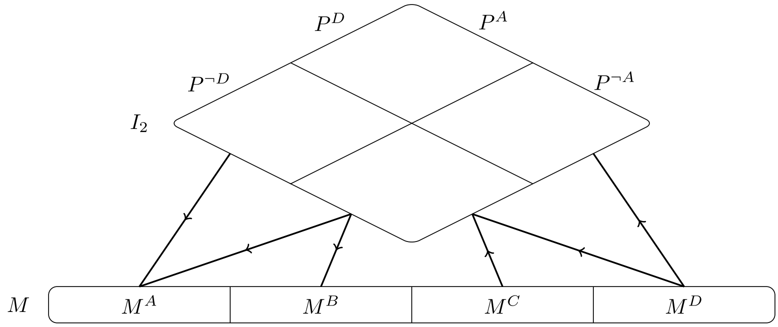


Fig. 4 An illustration of the sets P^{Δ} for $\Delta \in \{A, \neg A, \neg D, D\}$ and their relation with the sets M^{Γ} for $\Gamma \in \{A, B, C, D\}$

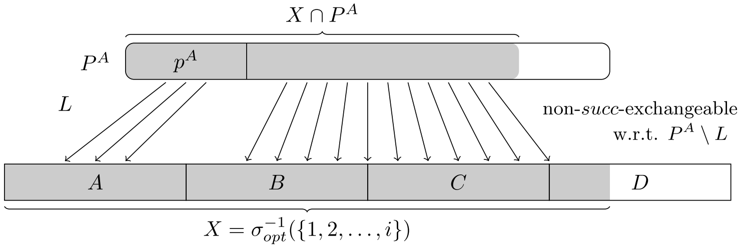


Fig. 5 An illustration of the proof of Lemma 2.9 for $(\Gamma, \Delta) = (A, A)$

non-*succ*-exchangeable with respect to $P^A \setminus X^A_A$; therefore, by Lemma 2.6, we can consider only a very narrow choice of X^A_{BCD}. Thus, the whole part $X \cap P^A$ can be represented by its subset of cardinality at most p^A plus some small information about the rest. If p^A is significantly smaller than $|P^A|/2$, this representation is more concise than simply remembering a subset of P^A. Thus we obtain a better bound on the number of feasible sets.

A symmetric situation arises when p^D is significantly smaller than $|P^D|/2$; moreover, we can similarly use Lemma 2.6 if p^B is significantly smaller than $|P^{\neg A}|/2$ or p^C than $|P^{\neg D}|/2$. This is formalized by the following lemma.

Lemma 2.9 *If* $p^{\Gamma} < |P^{\Delta}|/2$ *for some* $(\Gamma, \Delta) \in \{(A, A), (B, \neg A), (C, \neg D), (D, D)\}$ *and* $\varepsilon_1 \leq 1/4$, *then the DP algorithm can be augmented to solve the remaining instance in time bounded by*

$$T_p(n) = 2^{n-|P^{\Delta}|}\binom{|P^{\Delta}|}{p^{\Gamma}}\binom{n}{|M|}n^{O(1)}.$$

Proof We first describe in detail the case $\Delta = \Gamma = A$, and, later, we shortly describe the other cases that are proven analogously. An illustration of the proof is depicted on Fig. 5.

On a high-level, we want to proceed as in Proposition 2.1, i.e., use the standard DP algorithm described in Sect. 2.1, while terminating the computation for some unfeasible subsets of V. However, in this case we need to slightly modify the recursive formula used in the computations, and we compute $\sigma[X, L]$ for $X \subseteq V$, $L \subseteq X \cap P^A$. Intuitively, the set X plays the same role as before, whereas L is the subset of $X \cap P^A$ that was placed in the quarter A. Formally, $\sigma[X, L]$ is the ordering of X that attains the minimum total cost among those orderings σ for which $L = P^A \cap \sigma^{-1}(A)$. Thus, in the DP algorithm we use the following recursive formula:

$$T(\sigma[X, L]) = \begin{cases} \min_{v\in\max(X)}[T(\sigma[X\setminus\{v\}, L\setminus\{v\}]) + T(v, |X|)] \\ \qquad \text{if } |X| \le n/4 \text{ and } L = X \cap P^A, \\ +\infty \quad \text{if } |X| \le n/4 \text{ and } L \ne X \cap P^A, \\ \min_{v\in\max(X)\setminus L}[T(\sigma[X\setminus\{v\}, L]) + T(v, |X|)] \\ \qquad \text{otherwise.} \end{cases}$$

In the next paragraphs we describe a polynomial-time algorithm $\mathcal{R}$ that accepts or rejects pairs of subsets (X, L), $X \subseteq V$, $L \subseteq X \cap P^A$; we terminate the computation on rejected pairs (X, L). As each single calculation of $\sigma[X, L]$ uses at most $|X|$ recursive calls, the time complexity of the algorithm is bounded by the number of accepted pairs, up to a polynomial multiplicative factor. We now describe the algorithm $\mathcal{R}$.

First, given a pair (X, L), we ensure that we fulfill the guessed sets M^Γ and W^Γ_{half}, $\Gamma \in \{A, B, C, D\}$, that is: E.g., we require $M^B, W^B_{\text{half}} \subseteq X$ if $|X| \ge n/2$ and $(M^B \cup W^B_{\text{half}}) \cap X = \emptyset$ if $|X| \le n/4$. We require similar conditions for other quarters A, C and D. Moreover, we require that X is downward closed. Note that this implies $X \cap P^{\neg A} = \emptyset$ if $|X| \le n/4$ and $P^{\neg D} \subseteq X$ if $|X| \ge 3n/4$.

Second, we require the following:

1. If $|X| \le n/4$, we require that $L = X \cap P^A$ and $|L| \le p^A$; as $p^A \le |P^A|/2$, there are at most $2^{n-|P^A|}\binom{|P^A|}{p^A}n$ such pairs (X, L);
2. Otherwise, we require that $|L| = p^A$ and that the set $X \cap (P^A \setminus L)$ is non-*succ*-exchangeable with respect to $P^A \setminus L$; by Lemma 2.6 there are at most $\sum_{l\le|M|}\binom{|P^A\setminus L|}{l} \le n\binom{n}{|M|}$ (since $|M| \le 2\varepsilon_1 n \le n/2$) non-*succ*-exchangeable sets with respect to $P^A \setminus L$, thus there are at most $2^{n-|P^A|}\binom{|P^A|}{p^A}\binom{n}{|M|}n$ such pairs (X, L).

Let us now check the correctness of the above pruning. Let $0 \le i \le n$ and let $X = \sigma_{opt}^{-1}(\{1, 2, \dots, i\})$ and $L = \sigma_{opt}^{-1}(A) \cap X \cap P^A$. It is easy to see that Lemma 2.5 implies that in case $i \ge n/4$ the set $X \cap (P^A \setminus L)$ is non-*succ*-exchangeable and the pair (X, L) is accepted.

Let us now shortly discuss the case $\Gamma = B$ and $\Delta = \neg A$. Recall that, due to the precedence constraints between $P^{\neg A}$ and M^B, the jobs from $P^{\neg A}$ cannot be scheduled in the segment A. Therefore, while computing $\sigma[X]$ for $|X| \ge n/2$, we can represent $X \cap P^{\neg A}$ as a disjoint sum of two subsets $X^{\neg A}_B$, $X^{\neg A}_{CD}$: the first one, of size p^B, to be placed in B, and the second one to be placed in $C \cup D$. Recall that in Sect. 2.6 we have ensured that for any $v \in I_2$, all predecessors of v appear in M^{AB}

and all successors of v appear in M^{CD}. We infer that all predecessors of jobs in $X_{CD}^{\neg A}$ appear in segments A and B and, by Lemma 2.5, in the optimal solution the set $X_{CD}^{\neg A}$ is non-*succ*-exchangeable with respect to $P^{\neg A} \setminus X_B^{\neg A}$, Therefore we may proceed as in the case of $(\Gamma, \Delta) = (A, A)$; in particular, while computing $\sigma[X, L]$:

1. If $|X| \le n/4$, we require that $L = X \cap P^{\neg A} = \emptyset$;
2. If $n/4 < |X| \le n/2$, we require that $L = X \cap P^{\neg A}$ and $|L| \le p^B$;
3. Otherwise, we require that $|L| = p^B$ and that the set $X \cap (P^{\neg A} \setminus L)$ is non-*succ*-exchangeable with respect to $P^{\neg A} \setminus L$.

The cases $(\Gamma, \Delta) \in \{(C, \neg D), (D, D)\}$ are symmetrical: L corresponds to jobs from P^Δ scheduled to be done in segment Γ and we require that $X \cap (P^\Delta \setminus L)$ is non-*pred*-exchangeable (instead of non-*succ*-exchangeable) with respect to $P^\Delta \setminus L$. The recursive definition of $T(\sigma[X, L])$ should be also adjusted. ☐

Observe that if any of the sets P^Δ for $\Delta \in \{A, \neg A, \neg D, D\}$ is significantly larger than $n/2$ (i.e., larger than $(\frac{1}{2} + \delta)n$ for some $\delta > 0$), one of the situations in Lemma 2.9 indeed occurs, since $p^\Gamma \le n/4$ for $\Gamma \in \{A, B, C, D\}$ and $|M|$ is small.

Lemma 2.10 *If* $2\varepsilon_1 < 1/4 + \varepsilon_3/2$ *and at least one of the sets* P^A, $P^{\neg A}$, $P^{\neg D}$ *and* P^D *is of size at least* $(1/2 + \varepsilon_3)n$, *then the DP algorithm can be augmented to solve the remaining instance in time bounded by*

$$T_3(n) = 2^{(1/2-\varepsilon_3)n} \binom{(1/2 + \varepsilon_3)n}{n/4} \binom{n}{2\varepsilon_1 n} n^{O(1)}.$$

Proof The claim is straightforward; note only that the term $2^{n-|P^\Delta|}\binom{|P^\Delta|}{p^\Gamma}$ for $p^\Gamma < |P^\Delta|/2$ is a decreasing function of $|P^\Delta|$. ☐

Note that we have $10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} n^{O(1)}$ overhead so far. As $\binom{(1/2+\varepsilon_3)n}{n/4} = O((2 - c(\varepsilon_3))^{(1/2+\varepsilon_3)n})$ for some constant $c(\varepsilon_3) > 0$, for any small fixed ε_3 we can choose sufficiently small ε_2 and ε_1 to have $10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} n^{O(1)} T_3(n) = O(c^n)$ for some $c < 2$.

From this point we assume that $|P^A|, |P^{\neg A}|, |P^{\neg D}|, |P^D| \le (1/2 + \varepsilon_3)n$. As $P^A \cup P^{\neg A} = I_2 = P^{\neg D} \cup P^D$ and $|I_2| \ge (1 - 2\varepsilon_1 - 2\varepsilon_2)n$, this implies that these four sets are of size at least $(1/2 - 2\varepsilon_1 - 2\varepsilon_2 - \varepsilon_3)n$, i.e., they are of size roughly $n/2$. Having bounded the sizes of the sets P^Δ from below, we are able to use Lemma 2.9 again: if any of the numbers p^A, p^B, p^C, p^D is significantly smaller than $n/4$ (i.e., smaller than $(\frac{1}{4} - \delta)n$ for some $\delta > 0$), then it is also significantly smaller than half of the cardinality of the corresponding set P^Δ.

Lemma 2.11 *Let* $\varepsilon_{123} = 2\varepsilon_1 + 2\varepsilon_2 + \varepsilon_3$. *If at least one of the numbers* p^A, p^B, p^C *and* p^D *is smaller than* $(1/4 - \varepsilon_4)n$ *and* $\varepsilon_4 > \varepsilon_{123}/2$, *then the DP algorithm can be augmented to solve the remaining instance in time bounded by*

$$T_4(n) = 2^{(1/2+\varepsilon_{123})n} \binom{(1/2 - \varepsilon_{123})n}{(1/4 - \varepsilon_4)n} \binom{n}{2\varepsilon_1 n} n^{O(1)}.$$

Proof As, before, the claim is a straightforward application of Lemma 2.9, and the fact that the term $2^{n-|P^\Delta|}\binom{|P^\Delta|}{p^\Gamma}$ for $p^\Gamma < |P^\Delta|/2$ is a decreasing function of $|P^\Delta|$. □

So far we have $10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} n^{O(1)}$ overhead. Similarly as before, for any small fixed ε_4 if we choose $\varepsilon_1, \varepsilon_2, \varepsilon_3$ sufficiently small, we have $\binom{(1/2-\varepsilon_{123})n}{(1/4-\varepsilon_4)n} = O((2 - c(\varepsilon_4))^{(1/2-\varepsilon_{123})n})$ and $10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} n^{O(1)} T_4(n) = O(c^n)$ for some $c < 2$.

Thus we are left with the case when $p^A, p^B, p^C, p^D \geq (1/4 - \varepsilon_4)n$.

2.8 The Remaining Case

In this subsection we infer that in the remaining case the quarters A, B, C and D are somewhat independent, which allows us to develop a faster algorithm. More precisely, note that $p^\Gamma \geq (1/4 - \varepsilon_4)n$, $\Gamma \in \{A, B, C, D\}$, means that almost all elements that are placed in A by σ_{opt} belong to P^A, while almost all elements placed in B belong to $P^{\neg A}$. Similarly, almost all elements placed in D belong to P^D and almost all elements placed in C belong to $P^{\neg D}$. As $P^A \cap P^{\neg A} = \emptyset$ and $P^{\neg D} \cap P^D = \emptyset$, this implies that what happens in the quarters A and B, as well as C and D, is (almost) independent. This key observation can be used to develop an algorithm that solves this special case in time roughly $O(2^{3n/4})$.

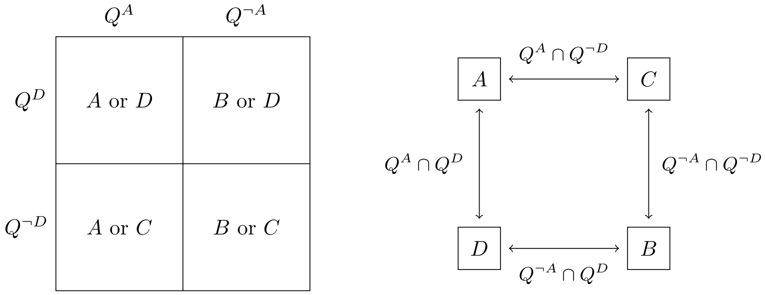
Let $W^B_{\text{quarter}} = I_2 \cap (\sigma^{-1}_{opt}(B) \setminus P^{\neg A})$ and $W^C_{\text{quarter}} = I_2 \cap (\sigma^{-1}_{opt}(C) \setminus P^{\neg D})$. As $p^B, p^C \geq (1/4 - \varepsilon_4)n$ we have that $|W^B_{\text{quarter}}|, |W^C_{\text{quarter}}| \leq \varepsilon_4 n$. We branch into at most $n^2\binom{n}{\varepsilon_4 n}^2$ subcases, guessing the sets W^B_{quarter} and W^C_{quarter}. Let $W_{\text{quarter}} = W^B_{\text{quarter}} \cup W^C_{\text{quarter}}$, $I_3 = I_2 \setminus W_{\text{quarter}}$, $Q^\Delta = P^\Delta \setminus W_{\text{quarter}}$ for $\Delta \in \{A, \neg A, \neg D, D\}$. Moreover, let $W^\Gamma = M^\Gamma \cup W^\Gamma_{\text{half}} \cup W^\Gamma_{\text{quarter}}$ for $\Gamma \in \{A, B, C, D\}$, using the convention $W^A_{\text{quarter}} = W^D_{\text{quarter}} = \emptyset$.

Note that in the current branch for any ordering and any $\Gamma \in \{A, B, C, D\}$, the segment Γ gets all the jobs from W^Γ and $q^\Gamma = n/4 - |W^\Gamma|$ jobs from appropriate Q^Δ ($\Delta = A, \neg A, \neg D, D$ for $\Gamma = A, B, C, D$, respectively). Thus, the behavior of an ordering σ in A influences the behavior of σ in C by the choice of which elements of $Q^A \cap Q^{\neg D}$ are placed in A, and which in C. Similar dependencies are between A and D, B and C, as well as B and D (see Fig. 6). In particular, there are no dependencies between A and B, as well as C and D, and we can compute the optimal arrangement by keeping track of only three out of four dependencies at once, leading us to an algorithm running in time roughly $O(2^{3n/4})$. This is formalized in the following lemma:

Lemma 2.12 *If* $2\varepsilon_1 + 2\varepsilon_2 + \varepsilon_4 < 1/4$ *and the assumptions of Lemmata 2.2 and 2.8–2.11 are not satisfied, the instance can be solved by an algorithm running in time bounded by*

$$T_5(n) = \binom{n}{\varepsilon_4 n}^2 2^{(3/4+\varepsilon_3)n} n^{O(1)}.$$

Proof Let $(\Gamma, \Delta) \in \{(A, A), (B, \neg A), (C, \neg D), (D, D)\}$. For each set $Y \subseteq Q^\Delta$ of size q^Γ, for each bijection (partial ordering) $\sigma^\Gamma(Y) : Y \cup W^\Gamma \to \Gamma$ let us define its



	Q^A	$Q^{\neg A}$
Q^D	A or D	B or D
$Q^{\neg D}$	A or C	B or C

Fig. 6 Dependencies between quarters and sets Q^Δ. The *left part* of the figure illustrates where the jobs from $Q^{\Delta_1} \cap Q^{\Delta_2}$ may be placed. The *right part* of the figure illustrates the dependencies between the quarters

cost as

$$T(\sigma^\Gamma(Y)) = \sum_{v \in Y \cup W^\Gamma} T(v, \sigma^\Gamma(Y)(v)).$$

Let $\sigma^\Gamma_{opt}(Y)$ be the partial ordering that minimizes the cost (recall that it is unique due to the initial steps in Sect. 2.4). Note that if we define $Y^\Gamma_{opt} = \sigma^{-1}_{opt}(\Gamma) \cap Q^\Delta$ for $(\Gamma, \Delta) \in \{(A, A), (B, \neg A), (C, \neg D), (D, D)\}$, then the ordering σ_{opt} consists of the partial orderings $\sigma^\Gamma_{opt}(Y^\Gamma_{opt})$.

We first compute the values $\sigma^\Gamma_{opt}(Y)$ for all $(\Gamma, \Delta) \in \{(A, A), (B, \neg A), (C, \neg D), (D, D)\}$ and $Y \subseteq Q^\Delta$, $|Y| = q^\Gamma$, by a straightforward modification of the DP algorithm. For fixed pair (Γ, Δ), the DP algorithm computes $\sigma^\Gamma_{opt}(Y)$ for all Y in time

$$2^{|W^\Gamma|+|Q^\Delta|} n^{O(1)} \le 2^{(2\varepsilon_1+2\varepsilon_2+\varepsilon_4)n+(1/2+\varepsilon_3)n} n^{O(1)} = O\big(2^{(3/4+\varepsilon_3)n}\big).$$

The last inequality follows from the assumption $2\varepsilon_1 + 2\varepsilon_2 + \varepsilon_4 < 1/4$.

Let us focus on the sets $Q^A \cap Q^{\neg D}$, $Q^A \cap Q^D$, $Q^{\neg A} \cap Q^{\neg D}$ and $Q^{\neg A} \cap Q^D$. Without loss of generality we assume that $Q^A \cap Q^{\neg D}$ is the smallest among those. As they all are pairwise disjoint and sum up to I_2, we have $|Q^A \cap Q^{\neg D}| \le n/4$. We branch into at most $2^{|Q^A \cap Q^{\neg D}|+|Q^{\neg A} \cap Q^D|}$ subcases, guessing the sets

$$Y^{AC}_{opt} = Y^A_{opt} \cap \big(Q^A \cap Q^{\neg D}\big) = \big(Q^A \cap Q^{\neg D}\big) \setminus Y^C_{opt} \quad \text{and}$$
$$Y^{BD}_{opt} = Y^B_{opt} \cap \big(Q^{\neg A} \cap Q^D\big) = \big(Q^{\neg A} \cap Q^D\big) \setminus Y^D_{opt}.$$

Then, we choose the set

$$Y^{AD}_{opt} = Y^A_{opt} \cap \big(Q^A \cap Q^D\big) = \big(Q^A \cap Q^D\big) \setminus Y^D_{opt}$$

that optimizes

$$T\big(\sigma^A_{opt}\big(Y^{AC}_{opt} \cup Y^{AD}_{opt}\big)\big) + T\big(\sigma^D_{opt}\big(Q^D \setminus \big(Y^{AD}_{opt} \cup Y^{BD}_{opt}\big)\big)\big).$$

Independently, we choose the set

$$Y^{BC}_{opt} = Y^B_{opt} \cap \big(Q^{\neg A} \cap Q^{\neg D}\big) = \big(Q^{\neg A} \cap Q^{\neg D}\big) \setminus Y^C_{opt}$$

Table 1 Summary of running times of all cases of the algorithm

Reference	Running time
Lemma 2.2	$T_1(n) = O^\star((3/4)^{\varepsilon_1 n} 2^n)$
Lemma 2.8	$3^{\varepsilon_1 n} T_2(n) n^{O(1)} = 3^{\varepsilon_1 n}(2^{(1-\varepsilon_2)n} + \binom{n}{(1/2-\varepsilon_2)n} + 2^{\varepsilon_2 n}\binom{(1-\varepsilon_2)n}{n/2}) n^{O(1)}$
Lemma 2.10	$10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} T_3(n) n^{O(1)} = 10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} 2^{(1/2-\varepsilon_3)n} \binom{(1/2+\varepsilon_3)n}{n/4} \binom{n}{2\varepsilon_1 n} n^{O(1)}$
Lemma 2.11	$10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} T_4(n) n^{O(1)} = 10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} 2^{(1/2+2\varepsilon_1+2\varepsilon_2+\varepsilon_3)n} \binom{(1/2-2\varepsilon_1-2\varepsilon_2-\varepsilon_3)n}{(1/4-\varepsilon_4)n} \binom{n}{2\varepsilon_1 n} n^{O(1)}$
Lemma 2.12	$10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} T_5(n) n^{O(1)} = 10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} \binom{n}{\varepsilon_4 n}^2 2^{(3/4+\varepsilon_3)n} n^{O(1)}$

that optimizes

$$T\left(\sigma^B_{opt}\left(Y^{BC}_{opt} \cup Y^{BD}_{opt}\right)\right) + T\left(\sigma^C_{opt}\left(Q^{\neg D} \setminus \left(Y^{BC}_{opt} \cup Y^{AC}_{opt}\right)\right)\right).$$

To see the correctness of the above step, note that $Y^A_{opt} = Y^{AC}_{opt} \cup Y^{AD}_{opt}$, and similarly for other quarters.

The time complexity of the above step is bounded by

$$\begin{aligned} & 2^{|Q^A \cap Q^{\neg D}| + |Q^{\neg A} \cap Q^D|}\left(2^{|Q^A \cap Q^D|} + 2^{|Q^{\neg A} \cap Q^{\neg D}|}\right) n^{O(1)} \\ &= 2^{|Q^A \cap Q^{\neg D}|}\left(2^{|Q^D|} + 2^{|Q^{\neg A}|}\right) n^{O(1)} \\ &\le 2^{(3/4+\varepsilon_3)n} n^{O(1)} \end{aligned}$$

and the bound $T_5(n)$ follows. □

So far we have $10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} n^{O(1)}$ overhead. For sufficiently small ε_4 we have $\binom{n}{\varepsilon_4 n} = O(2^{n/16})$ and then for sufficiently small constants ε_k, $k = 1, 2, 3$ we have $10^{\varepsilon_1 n} 2^{2\varepsilon_2 n} n^{O(1)} T_5(n) = O(c^n)$ for some $c < 2$.

2.9 Numerical Values of the Constants

Table 1 summarizes the running times of all cases of the algorithm. Using the following values of the constants:

$$\begin{aligned} \varepsilon_1 &= 2.677001953125 \cdot 10^{-10}, \\ \varepsilon_2 &= 0.0000272462885123491287231445312 5, \\ \varepsilon_3 &= 0.007010121770270753069780766963958740234375, \\ \varepsilon_4 &= 0.01652675350589504740935353765962645411491394042968 8 \end{aligned}$$

we get that the running time of our algorithm is bounded by:

$$O\left(\left(2 - 10^{-10}\right)^n\right).$$

3 Conclusion

We presented an algorithm that solves SCHED in $O((2-\varepsilon)^n)$ time for some small ε. This shows that in some sense SCHED appears to be easier than resolving CNF-SAT formulae, which is conjectured to need 2^n time (the so-called Strong Exponential Time Hypothesis). Our algorithm is based on an interesting property of the optimal solution expressed in Lemma 2.6, which can be of independent interest. However, our best efforts to numerically compute an optimal choice of values of the constants ε_k, $k = 1, 2, 3, 4$ lead us to an ε of the order of 10^{-10}. Although Lemma 2.6 seems powerful, we lost a lot while applying it. In particular, the worst trade-off seems to happen in Sect. 2.6, where ε_1 needs to be chosen much smaller than ε_2. The natural question is: can the base of the exponent be significantly improved?

Acknowledgements We thank Dominik Scheder for very useful discussions on the SCHED problem during his stay in Warsaw. Moreover, we greatly appreciate the detailed comments of anonymous reviewers, especially regarding presentation issues and minor optimizations in our algorithm.

References

1. Binkele-Raible, D., Brankovic, L., Cygan, M., Fernau, H., Kneis, J., Kratsch, D., Langer, A., Liedloff, M., Pilipczuk, M., Rossmanith, P., Wojtaszczyk, J.O.: Breaking the 2^n-barrier for irredundance: Two lines of attack. J. Discrete Algorithms **9**(3), 214–230 (2011)
2. Björklund, A.: Determinant sums for undirected hamiltonicity. In: 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 173–182. IEEE Comput. Soc., Los Alamitos (2010)
3. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: fast subset convolution. In: 39th Annual ACM Symposium on Theory of Computing (STOC), pp. 67–74 (2007). doi:10.1145/1250790.1250801
4. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion–exclusion. SIAM J. Comput. **39**(2), 546–563 (2009)
5. Brucker, P.: Scheduling Algorithms, 2nd edn. Springer, Heidelberg (1998)
6. Chekuri, C., Motwani, R.: Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. Discrete Appl. Math. **98**(1-2), 29–38 (1999)
7. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: 52nd Annual Symposium on Foundations of Computer Science (FOCS), pp. 150–159. IEEE Press, New York (2011)
8. Cygan, M., Pilipczuk, M.: Exact and approximate bandwidth. Theor. Comput. Sci. **411**(40-42), 3701–3713 (2010)
9. Cygan, M., Pilipczuk, M., Wojtaszczyk, J.O.: Capacitated domination faster than $O(2^n)$. Inf. Process. Lett. **111**, 1099–1103 (2011)
10. Fomin, F., Kratsch, D.: Exact Exponential Algorithms. Springer, Berlin (2010)
11. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. ACM **56**(5), 1–32 (2009). doi:10.1145/1552285.1552286
12. Fomin, F.V., Todinca, I., Villanger, Y.: Exact algorithm for the maximum induced planar subgraph problem. In: 19th Annual European Symposium on Algorithms (ESA). Lecture Notes in Computer Science, vol. 6942, pp. 287–298. Springer, Berlin (2011)
13. Graham, R.: Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. **45**, 1563–1581 (1966)
14. Hefetz, N., Adiri, I.: An efficient optimal algorithm for the two-machines unit-time jobshop schedule-length problem. Math. Oper. Res. **7**, 354–360 (1982)
15. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. **62**(2), 367–375 (2001)

16. Lawler, E.L.: Optimal sequencing of a single machine subject to precedence constraints. Manag. Sci. **19**, 544–546 (1973)
17. Lenstra, J.K., Kan, A.H.G.R., Brucker, P.: Complexity of machine scheduling problems. Ann. Discrete Math. **1**, 343–362 (1977)
18. Lenstra, J.K., Kan, A.H.G.R.: Complexity of scheduling under precedence constraints. Oper. Res. **26**, 22–35 (1978)
19. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs of bounded treewidth are probably optimal. In: Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 777–789 (2011)
20. Margot, F., Queyranne, M., Wang, Y.: Decompositions, network flows, and a precedence constrained single-machine scheduling problem. Oper. Res. **51**(6), 981–992 (2003)
21. Mucha, M., Sankowski, P.: Maximum matchings via Gaussian elimination. In: 45th Symposium on Foundations of Computer Science (FOCS), pp. 248–255. IEEE Comput. Soc., Los Alamitos (2004)
22. Patrascu, M., Williams, R.: On the possibility of faster SAT algorithms. In: Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1065–1075 (2010)
23. van Rooij, J.M.M., Nederlof, J., van Dijk, T.C.: Inclusion/exclusion meets measure and conquer. In: 17th Annual European Symposium (ESA). Lecture Notes in Computer Science, vol. 5757, pp. 554–565. Springer, Berlin (2009)
24. Woeginger, G.J.: Space and time complexity of exact algorithms: Some open problems (invited talk). In: First International Workshop on Parameterized and Exact Computation (IWPEC). Lecture Notes in Computer Science, vol. 3162, pp. 281–290. Springer, Berlin (2004)
25. Woeginger, G.J.: Open problems around exact algorithms. Discrete Appl. Math. **156**(3), 397–405 (2008)