Scheduling partially ordered jobs faster than 2^{n*}

Marek Cygan[†] Marcin Pilipczuk[‡] Michał Pilipczuk[§] Jakub Onufry Wojtaszczyk[¶]

Abstract

In the SCHED problem 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. SCHED 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 SCHED 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.

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. The question asked in the moderately exponential time algorithms area is how small the constant c in the c^n time algorithms can be. Many difficult problems can be solved much faster than by the obvious brute-force algorithm; examples are Independent Set [10], Dominating Set [10, 17], Chromatic Number [4] and Bandwidth [7]. The race for the fastest exact algorithm inspired several very interesting tools and techniques such as Fast Subset Convolution [3] and Measure&Conquer [10] (for an overview of the field we refer the reader to a recent book by Fomin and Kratsch [9]).

For several problems, including TSP, Chromatic Number, Permanent, Set Cover, #Hamiltonian Cycles and SAT, the currently best known time complexity is of the form $O(2^n n^{O(1)})$, 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 [11], which is used as an argument that other problems are hard [6, 13, 16]. Recently, on the positive side, $O(c^n)$ time algorithms for a constant $c < 2$ have been developed for Capacitated Domination [8], Irredundance [1] and (a major breakthrough in the field) for the undirected version of the Hamiltonian Cycle problem [2].

In this paper we study the SCHED problem, defined as follows.

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

[†]Institute of Informatics, University of Warsaw, Poland, cygan@mimuw.edu.pl. Supported by Polish Ministry of Science grant no. N206 355636 and Foundation for Polish Science.

[‡]Institute of Informatics, University of Warsaw, Poland, malcin@mimuw.edu.pl. Supported by Polish Ministry of Science grant no. N206 355636 and Foundation for Polish Science.

[§]Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland, michal.pilipczuk@students.mimuw.edu.pl

[¶]Google Inc., Cracow, Poland, onufry@google.com

SCHED
Input: A partially ordered set $(V, \leq)$, (the elements of which are called *jobs*) together with a nonnegative processing time $t(v)$ for each job $v \in V$.
Task: Compute a bijection $\sigma : V \to \{1, 2, \ldots, |V|\}$ (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

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

If $u < v$ for $u, v \in V$ (i.e., $u \leq v$ and $u \neq v$), we say that u *precedes* v, or that u is *required* for v. We denote $|V|$ by n. SCHED is a special case of the precedence constrained Travelling Repairman Problem (prec-TRP), which is a relative of TSP. SCHED was shown to be NP-complete in 1978 by Lenstra and Rinnooy Kan [12], whereas to the best of our knowledge the currently smallest approximation ratio equals 2, due to independently discovered algorithms by Chekuri and Motwani [5] as well as Margot et al. [14].

Woeginger at IWPEC 2004 [18] has posed the question (repeated in 2008 [19]), whether it is possible to construct an $O((2-\varepsilon)^n)$ time algorithm for the SCHED problem. In this paper we present such an algorithm, thus affirmatively answering Woeginger's question. This result is intriguing in particular because the SCHED problem admits arbitrary processing times — and still it is possible to obtain an $O((2-\varepsilon)^n)$ time algorithm nonetheless. Probably due to arbitrary processing times Woeginger also asked [18, 19] whether an $O((2-\varepsilon)^n)$ time algorithm for one of the problems TRP, TSP, prec-TSP, SCHED implies $O((2-\varepsilon)^n)$ time algorithms for the other problems. This problem is still open. One should note that Woeginger in his papers asks for an $O(1.99^n)$ algorithm, though the intention is clearly to ask for an $O((2-\varepsilon)^n)$ algorithm.

The most important ingredient of our algorithm is a combinatorial Lemma (Lemma 2.5) 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 $(2-\varepsilon)^n$ 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

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

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[1], 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 $\sigma[X] : X \to \{1, 2, \ldots, |X|\}$ 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:

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

[1] By $O^\star()$ we denote the standard big O notation where we suppress polynomially bounded terms.

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 will be no 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 folowing 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 $c^{|V|}$ for some constant c. Then one can in time $O^\star(c^n)$ find an optimal ordering of the jobs in V among those orderings σ where $\sigma^{-1}(\{1, 2, \ldots, i\})$ is accepted by $\mathcal{R}$ for all $1 \leq i \leq n$.*

Proof. As discussed before, calculate $\sigma[V]$ recursively, using formula (1), storing all computed values $\sigma[X]$ in memory to avoid recomputation. Whenever we access a value $T(\sigma[X])$ for a set X not accepted by $\mathcal{R}$, we take $T(\sigma[X]) = \infty$. As each application of the formula (1) gives at most n recursive calls, the 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 M be a maximum matching[2] in G, which can be found in polynomial time [15]. If $X \subseteq V$ is downward closed, and $uv \in M$, $u < v$, 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 $2^{n-2|M|}3^{|M|}$. If $|M| \geq \varepsilon_1 n$, then we can solve the* SCHED *problem in time*

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

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 $|M| < \varepsilon_1 n$.

Let us fix a maximum matching M, let $W_1 \subseteq V$ be the set of endpoints of M, and let $I_1 = V \setminus W_1$. Note that, as M is a maximum matching in G, no two jobs in I_1 are bound by a precedence constraint, while $|W_1| \leq 2\varepsilon_1 n$, $|I_1| \geq (1 - 2\varepsilon_1)n$.

2.3 High-level overview — part 2

We are left in the situation where there is a small number of "special" elements (W_1), and the bulk remainder (I_1), consisting of elements that are tied by precedence constraints only to W_1 and not to each other.

First notice that if W_1 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 $w \in W_1$ would be of the form $u < w$ — that is, if the jobs from I_1 had no prerequisites. For any prefix set candidate $\tilde{X}$ we consider $X = \tilde{X} \cap I_1$. Now for any $x \in X$,

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

$y \in I_1 \setminus X$ we have an alternative prefix candidate: the set $\tilde{X}' = (\tilde{X} \cup \{y\}) \setminus \{x\}$. If $t(y) < t(x)$, there has to be a reason why $\tilde{X}'$ is not a strictly better prefix candidate than $\tilde{X}$ — namely, there has to exist $w \in W_1$ such that $x < w$, but $y \not< w$.

A similar reasoning would hold even if not all of I_1 had no perquisite's, but just some significant fraction J of I — again, the only feasible prefix candidates would be those in which for every $x \in X \cap J$ and $y \in J \setminus X$ there is a reason (either $t(x) < t(y)$ or an element $w \in W_1$ 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 a feasible set with J is significantly smaller than $2^{|J|}$. This is formalized and proved in Lemma 2.5, and is the cornerstone of the whole paper.

The 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 requisites of elements of K appear on positions j and earlier. If K is large (a constant fraction of n), this will be enough to limit the number of feasible sets to $(2-\varepsilon)^n$. The reasoning is to show there are significantly less 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 2.5 — and when we do the calculations, it turns out the resulting number of possibilities is significantly 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 W_1. We achieve this by branching into $4^{|W_1|}$ cases, for each element $w \in W_1$ choosing to which of the four quarters of the set $\{1, \ldots, n\}$ will $\sigma_{opt}(w)$ belong. This incurs a multiplicative cost of $4^{|W_1|}$, which will be offset by the gains from applying Lemma 2.5.

We will now repeatedly apply Lemma 2.5 to obtain information about the positions of various elements of I_1. We will repeatedly say that if "many" elements (by which 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 significantly over $n/2$ of them, we can apply the above reasoning for $j = n/4$ (Lemma 2.9). Subsequent lemmata bound the number of feasible sets if there are many elements that cannot appear in any of the two first quarters (Lemma 2.7), if significantly *less* than $n/2$ elements can appear in the first quarter (Lemma 2.9) and if a significant number of elements in the second quarter could actually appear in the first quarter (Lemma 2.10). 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 2.11).

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 significantly 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 Section 3, 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 $\pi : V \to \{1, 2, \ldots, n\}$ 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 W_1 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 W_1.

2.5 The core lemma

We now formalize the idea of exchanges presented in Section 2.3. In the proof of the first case we exchange u with some v_w whereas in the second case we exchange v with some u_w.

Definition 2.3. *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 (non)-exchangeable.

The applicability of this definition is in the following observation:

Observation 2.4. Let $K \subseteq I_1$. If for any $v \in K, w \in pred(K)$ we have that $\sigma_{opt}(v) > \sigma_{opt}(w)$, then for any $1 \leq i \leq n$ the set $K \cap \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ is non-*succ*-exchangeable with respect to K.

Similarly, if for any $v \in K, w \in succ(K)$ we have $\sigma_{opt}(v) < \sigma_{opt}(w)$, then the sets $K \cap \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ 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 Figure 1 for an illustration on the *succ*-exchangeable case.

Non-*succ*-exchangable sets. Assume, by contradiction, that for some i the set $L = K \cap \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ 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 2.3, 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 Observation. If $u < y$, then $\sigma'(y) = \sigma_{opt}(y) \geq \sigma_{opt}(w) > \sigma_{opt}(v_w) = \sigma'(u)$, 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*-exchangable sets. Assume, by contradiction, that for some i the set $L = K \cap \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ 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 2.3, 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 Observation. 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. □

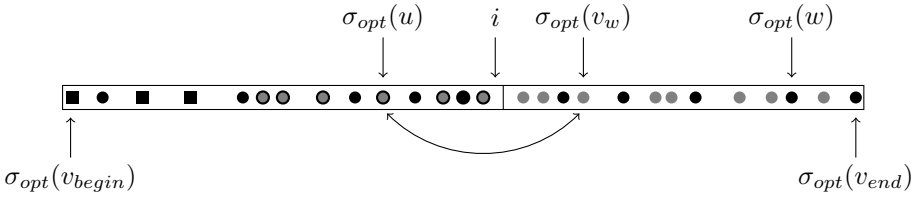

Figure 1: Figure illustrating the *succ*-exchangeable case of Observation 2.4. Gray circles indicate positions of elements of K, black contour indicates that an element is also in L. Black squares indicate positions of elements from $pred(K)$, and black circles — positions of other elements from W_1.

This means that if we manage to identify a set K satisfying the assumptions of the observation, 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, and we can identify them easily.

Lemma 2.5. *For any set $K \subseteq I_1$ the number of non-succ-exchangeable (non-pred-exchangeable) subsets is at most $\sum_{l \leq |W_1|} \binom{|K|}{l}$. 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 W_1. 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 Observation 2.4, 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 $f_Y : W_1 \to K \cup \{\text{nil}\}$ as follows: for any element $w \in W_1$ 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 *least expensive predecessors outside Y*) to be the set $\{f_Y(w) : w \in W_1\} \setminus \{\text{nil}\}$. $f(Y)$ is indeed a set of cardinality at most $|W_1|$, we will 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 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 $Y \subseteq K$. Indeed — 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 $u \in Y \setminus g(f(Y))$. We want to show that Y is *succ*-exchangeable. Consider any $w \in succ(u)$. As $u \notin g(f(Y))$,

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 $f_Y : W_1 \to K \cup \{\text{nil}\}$ as follows: for any element $w \in W_1$ 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 *most expensive successors in Y*) to be the set $\{f_Y(w) : w \in W_1\} \setminus \{\text{nil}\}$. $f(Y)$ is indeed a set of cardinality at most $|W_1|$, we will 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 $w \in pred(v)$ there exists a $z_w \in Z \cap succ(w)$ 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 $w \in pred(v)$.

First we prove $Y \subseteq g(f(Y))$ for any $Y \subseteq K$. Indeed — take any $v \in Y$ and consider any $w \in pred(v)$. Then $f_Y(w) \neq \text{nil}$ and $t(f_Y(w)) \geq t(v)$, as $v \in Y \cap succ(w)$. Thus $v \in g(f(Y))$, as for any $w \in pred(v)$ 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 $v \in g(f(Y)) \setminus Y$. We want to show that Y is *pred*-exchangeable. Consider any $w \in pred(v)$. As $v \in g(f(Y))$ there exists $z_w \in f(Y) \cap succ(w)$ 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 $\sum_{l=0}^{|W_1|} \binom{|K|}{l}$ 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 $O(|K|^2|W_1|)$ directly from the definition. □

Example 2.6. To illustrate the applicability of Lemma 2.5, we analyze the following very simple case: assume the whole set $W_1 \setminus \{v_{begin}\}$ succeeds I_1, i.e., for every $w \in W_1 \setminus \{v_{begin}\}$ and $v \in I_1$, if w and v are bound by some precedence constraint, then $v < w$. If ε_1 is small, then we can use the first case of Observation 2.4 for the whole set $K = I_1$: we have $pred(K) = \{v_{begin}\}$ 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 2.5, the number of sets accepted by $\mathcal{R}$ is bounded by $2^{|W_1|} \sum_{l \leq |W_1|} \binom{|I_1|}{l}$, which is small if $|W_1| \leq \varepsilon_1 n$.

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 W_1 in order to distinguish a suitable place for an application. As $|W_1| \leq 2\varepsilon_1 n$, we can proceed with quite an extensive branching on W_1.

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 $w \in W_1 \setminus \{v_{begin}, v_{end}\}$ we branch into four cases: whether $\sigma_{opt}(w)$ belongs to A, B, C or D. This branching leads to $4^{|W_1|-2} \leq 2^{4\varepsilon_1 n}$ subcases, and thus the same overhead in the time complexity. Of course, we already know that $\sigma_{opt}(v_{begin}) \in A$ and $\sigma_{opt}(v_{end}) \in D$. We terminate all the branches, where the guesses about alignment of jobs from W_1 contradict precedence constraints inside W_1.

In a fixed branch, let W_1^Γ be the set of elements of W_1 to be placed in Γ, for $\Gamma \in \{A, B, C, D\}$. Moreover let $W_1^{AB} = W_1^A \cup W_1^B$ and $W_1^{CD} = W_1^C \cup W_1^D$.

Let us now see what we can learn from the above step about the behaviour of σ_{opt} on I_1. Let

$$W_2^{AB} = \left\{v \in I_1 : \exists_w \ \left(w \in W_1^{AB} \wedge v < w\right)\right\}$$
$$W_2^{CD} = \left\{v \in I_1 : \exists_w \ \left(w \in W_1^{CD} \wedge w < v\right)\right\},$$

that is W_2^{AB} (resp. W_2^{CD}) are those elements of I_1 which are forced into the first (resp. second) half of σ_{opt} by the choices we made about W_1. If one of the W_2 sets is significantly larger than W_1, we have obtained a gain — by branching into $2^{4\varepsilon_1 n}$ branches we gained additional information about a significant 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 2.7. *Let $0 < \alpha < 1/2$ be a fixed constant. If W_2^{AB} or W_2^{CD} has at least $\varepsilon_2 n$ elements, then the DP algorithm can be augmented to solve the remaining instance in time*

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

Proof. We describe here only the case $|W_2^{AB}| \geq \varepsilon_2 n$. The second case is symmetrical.

Recall that the set W_2^{AB} needs to be placed in $A \cup B$ by the optimal ordering σ_{opt}. We use Proposition 2.1 using an algorithm $\mathcal{R}$ that accepts the following sets $X \subseteq V$:

1. all sets X of size at most $n/2 - \alpha|W_2^{AB}|$, there are at most $\binom{n}{(1/2-\alpha\varepsilon_2)n} n$ such sets;
2. among sets X of size $n/2 - \alpha|W_2^{AB}| \leq |X| \leq n/2$, only those sets for which $|W_2^{AB} \setminus X| \leq \alpha|W_2^{AB}|$, there are at most $2^{(1-\varepsilon_2)n}\binom{\varepsilon_2 n}{\alpha\varepsilon_2 \cdot n} n$ such sets;
3. among sets X of size at least $n/2$, only the sets containing W_2^{AB}, there are at most $2^{(1-\varepsilon_2)n}$ such sets.

Moreover, the algorithm $\mathcal{R}$ tests, if the set X conforms with the guessed sets W_1^Γ for $\Gamma \in \{A, B, C, D\}$, i.e.:

$$\begin{aligned}
|X| \leq n/4 &\Rightarrow (W_1^B \cup W_1^C \cup W_1^D) \cap X = \emptyset \\
n/4 \leq |X| \leq n/2 &\Rightarrow (W_1^A \subseteq X \wedge (W_1^C \cup W_1^D) \cap X = \emptyset) \\
n/2 \leq |X| \leq 3n/4 &\Rightarrow ((W_1^A \cup W_1^B) \subseteq X \wedge W_1^D \cap X = \emptyset) \\
3n/4 \leq |X| &\Rightarrow (W_1^A \cup W_1^B \cup W_1^C) \subseteq X.
\end{aligned}$$

To see that $\sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ is accepted by $\mathcal{R}$ for any $1 \leq i \leq n$, note that since W_2^{AB} is placed in $A \cup B$ by σ_{opt}, then for $i \leq n/2$ we have

$$|W_2^{AB} \setminus \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})| \leq n/2 - i.$$

The bound $T_2(n)$ is immediate from the discussion above. □

Note that we have $2^{4\varepsilon_1 n}$ overhead so far, due to guessing placement of the jobs from W_1. As $\binom{\varepsilon_2 n}{\alpha\varepsilon_2 n} = O((2 - c(\alpha))^{\varepsilon_2 n})$ and $\binom{n}{(1/2-\alpha\varepsilon_2)n} = O((2 - c(\alpha, \varepsilon_2))^n)$, for any small fixed ε_2 and any fixed $0 < \alpha < 1/2$ we can choose ε_1 sufficiently small so that $2^{4\varepsilon_1 n} T_2(n) = O(c^n)$ for some $c < 2$. Note that $2^{4\varepsilon_1 n} T_2(n)$ is an upper bound on the total time spent on processing all the considered subcases.

Let $W_2 = W_2^{AB} \cup W_2^{CD}$ and $I_2 = I_1 \setminus W_2$. From this point we assume that $|W_2^{AB}|, |W_2^{CD}| \leq \varepsilon_2 n$, hence $|W_2| \leq 2\varepsilon_2 n$ and $|I_2| \geq (1 - 2\varepsilon_1 - 2\varepsilon_2)n$. For each $v \in W_2^{AB}$ we branch into two subcases, whether $\sigma_{opt}(v)$ belongs to A or B. Similarly, for each $v \in W_2^{CD}$ we guess whether $\sigma_{opt}(v)$ belongs to C or D. Again, we execute only branches which are not trivially contradicting the constraints. This steps gives us $2^{|W_2|} \leq 2^{2\varepsilon_2 n}$ overhead in the time complexity. We denote the set of elements of W_2 assigned to quarter $\Gamma \in \{A, B, C, D\}$ by W_2^Γ.

2.7 Quarters and applications of the core lemma

In this section we try to apply Lemma 2.5 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 2.5 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 more detailed argumentation.

We define the following two partitions of I_2:

$$\begin{aligned} P^{\neg A} &= \left\{v \in I_2 : \exists_w \left(w \in W_1^B \wedge w < v\right)\right\}, \\ P^A &= I_2 \setminus P^{\neg A} = \left\{v \in I_2 : \forall_w \left(w < v \Rightarrow w \in W_1^A\right)\right\}, \\ P^{\neg D} &= \left\{v \in I_2 : \exists_w \left(w \in W_1^C \wedge w > v\right)\right\}, \\ P^D &= I_2 \setminus P^{\neg D} = \left\{v \in I_2 : \forall_w \left(w > v \Rightarrow w \in W_1^D\right)\right\}. \end{aligned}$$

In other words, the elements of $P^{\neg A}$ cannot be placed in A because some of their requirements are in W_1^B, and the elements of $P^{\neg D}$ cannot be placed in D because they are required by some elements of W_1^C. 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

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

Note that $p^\Gamma \leq n/4$ for every $\Gamma \in \{A, B, C, D\}$. As $p^A = n/4 - |W_1^A \cup W_2^A|$, $p^D = n/4 - |W_1^D \cup W_2^D|$, these values can be computed by the algorithm. We branch into $(1 + n/4)^2$ 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 $|P^A|/2$. We claim that we can apply Lemma 2.5 as follows. While computing $\sigma[X]$, if $|X| \geq n/4$, we can represent $X \cap P^A$ as a disjoint sum of two subsets $X_A^A, X_{BCD}^A \subseteq P^A$. 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_{BCD}^A have all predecessors in the quarter A, so by Observation 2.4 the set X_{BCD}^A has to be non-*succ*-exchangeable with respect to $X \cap P^A$; therefore, we can consider only a very narrow choice of X_{BCD}^A. 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 2.5 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 2.8. *If $p^\Gamma < |P^\Delta|/2$ for some $(\Gamma, \Delta) \in \{(A, A), (B, \neg A), (C, \neg D), (D, D)\}$ and $|P^\Delta| > 2|W_1|$, then the DP algorithm can be augmented to solve the remaining instance in time bounded by*

$$T_p(n) = 2^{n-|P^\Delta|} \binom{|P^\Delta|}{p^\Gamma} \binom{n}{|W_1|} n^{O(1)}.$$

Proof. We describe here only the case $\Delta = \Gamma = A$, the other cases are analogous.

On high-level, we want to proceed as in Proposition 2.1, i.e., use the standard DP algorithm described in Section 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:

$$T(\sigma[X, L]) = \begin{cases} \min_{v \in \max(X) \setminus (P^A \setminus L)} \left[T(\sigma[X \setminus \{v\}, L \setminus \{v\}]) + T(v, |X|)\right] & \text{if } |X| \leq n/4, \\ \min_{v \in \max(X) \setminus L} \left[T(\sigma[X \setminus \{v\}, L]) + T(v, |X|)\right] & \text{otherwise.} \end{cases}$$

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 $|X|$ 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 W_k^Γ, $\Gamma \in \{A, B, C, D\}$, $k = 1, 2$. E.g., we require $W_k^B \subseteq X$ if $|X| \geq n/2$ and $W_k^B \cap X = \emptyset$ if $|X| \leq n/4$. We require similar conditions for other quarters A, C and D (cf. proof of Lemma 2.7). Moreover, we require that X is downward closed. Note that this implies $X \cap P^{\neg A} = \emptyset$ if $|X| \leq n/4$ and $P^{\neg D} \subseteq X$ if $|X| \geq 3n/4$.

Second, we require the following:

1. If $|X| \leq n/4$, we require that $L = X \cap P^A$ and $|L| \leq p^A$; as $p^A \leq |P^A|/2$, 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 2.5 there are at most $2^{n-|P^A|}\binom{|P^A|}{p^A}\binom{n}{|W_1|} n$ such pairs (X, L).

Let us now check the correctness of the above pruning. Let $0 \leq i \leq n$ and let $X = \sigma_{opt}^{-1}(\{1, 2, \ldots, i\})$ and $L = \sigma_{opt}^{-1}(A) \cap X \cap P^A$. It is easy to see that Observation 2.4 implies that in case $i \geq n/4$ the set $X \cap (P^A \setminus L)$ is non-*succ*-exchangeable and the pair (X, L) is accepted.

The cases $(\Gamma, \Delta) \in \{(B, \neg A), (C, \neg D), (D, D)\}$ are analogous: 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) in case $\Delta = \neg D, D$. 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 $n/2$, one of the situations in Lemma 2.8 indeed occurs, since $p^\Gamma \leq n/4$ for $\Gamma \in \{A, B, C, D\}$.

Lemma 2.9. *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 $2^{(4\varepsilon_1+2\varepsilon_2)n} n^{O(1)}$ 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 $2^{(4\varepsilon_1+2\varepsilon_2)n} n^{O(1)} T_3(n) = O(c^n)$ for some $c < 2$.

From this point we assume that $|P^A|, |P^{\neg A}|, |P^{\neg D}|, |P^D| \leq (1/2 + \varepsilon_3)n$. As $P^A \cup P^{\neg A} = I_2 = P^{\neg D} \cup P^D$ and $|I_2| \geq (1 - 2\varepsilon_1 - 2\varepsilon_2)n$, this implies that all these sets are of size at least $(1/2 - 2\varepsilon_1 - 2\varepsilon_2 - \varepsilon_3)n$, i.e., there are of size roughly $n/2$. Having bounded the sizes of the sets P^Δ from below, we are able to use Lemma 2.8 again: if any of the numbers p^A, p^B, p^C, p^D is significantly smaller than $n/4$, then it is also significantly smaller than half of the cardinality of the corresponding set P^Δ.

Lemma 2.10. *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 straighforward application of Lemma 2.8, 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 $2^{(4\varepsilon_1+2\varepsilon_2)n}n^{O(1)}$ 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 $2^{(4\varepsilon_1+2\varepsilon_2)n}n^{O(1)}T_4(n) = O(c^n)$ 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 by σ_{opt} in A 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 $W_3^B = I_2 \cap (\sigma_{opt}^{-1}(B) \setminus P^{\neg A})$ and $W_3^C = I_2 \cap (\sigma_{opt}^{-1}(C) \setminus P^{\neg D})$. As $p^B, p^C \geq (1/4-\varepsilon_4)n$ we have that $|W_3^B|, |W_3^C| \leq \varepsilon_4 n$. We branch into at most $n^2\binom{n}{\varepsilon_4 n}^2$ subcases, guessing the sets W_3^B and W_3^C. Let $W_3 = W_3^B \cup W_3^C$, $I_3 = I_2 \setminus W_3$, $Q^\Delta = P^\Delta \setminus W_3$ for $\Delta \in \{A, \neg A, \neg D, D\}$. Moreover, let $W^\Gamma = W_1^\Gamma \cup W_2^\Gamma \cup W_3^\Gamma$ for $\Gamma \in \{A, B, C, D\}$, using the convention $W_3^A = W_3^D = \emptyset$.

Note that in the current branch any ordering puts into the segment Γ for $\Gamma \in \{A, B, C, D\}$ 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 behaviour of an ordering σ in A influences the behaviour 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. Thus, the dependencies form a 4-cycle, 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 2.11. *If $2\varepsilon_1 + 2\varepsilon_2 + \varepsilon_4 < 1/4$, the remaining case 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 cost as

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

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

We first compute the values $\sigma_{opt}^\Gamma(Y)$ 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 $\sigma_{opt}^\Gamma(Y)$ for all Y in time

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

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 $|Q^A \cap Q^{\neg D}| \leq n/4$. We branch into at most $2^{|Q^A \cap Q^{\neg D}| + |Q^{\neg A} \cap Q^D|}$ subcases, guessing the sets

$$Y_{opt}^{AC} = Y_{opt}^A \cap (Q^A \cap Q^{\neg D}) = (Q^A \cap Q^{\neg D}) \setminus Y_{opt}^C \quad \text{and}$$
$$Y_{opt}^{BD} = Y_{opt}^B \cap (Q^{\neg A} \cap Q^D) = (Q^{\neg A} \cap Q^D) \setminus Y_{opt}^D.$$

Then, we choose the set

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

that optimizes

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

Independently, we choose the set

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

that optimizes

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

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

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

and the bound $T_5(n)$ follows. □

So far we have $2^{(4\varepsilon_1+2\varepsilon_2)n} n^{O(1)}$ 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 $2^{(4\varepsilon_1+2\varepsilon_2)n} n^{O(1)} T_5(n) = O(c^n)$ for some $c < 2$.

3 Numerical values of the constants

Before we give a set of values of the constants ε_k and α (used in Lemma 2.7), let us make a small optimization. Note that when invoking Lemma 2.7, we only need to know how σ_{opt} splits the set W_1 between halves $A \cup B$ and $C \cup D$, i.e., we need to know the sets W_1^{AB} and W_1^{CD} instead of the whole quadruple W_1^Γ for $\Gamma \in \{A, B, C, D\}$. This leads to $2^{2\varepsilon_1 n}$ overhead (instead of $2^{4\varepsilon_1 n}$) in front of the bound $T_2(n)$. Using this observation and the following values of the constants:

$$\begin{aligned} \varepsilon_1 &= 9.98046875 \cdot 10^{-16} \\ \varepsilon_2 &= 0.00002246093750077387619018554687 5 \\ \varepsilon_3 &= 0.00701843070983939668794721364974975585937 5 \\ \varepsilon_4 &= 0.01652666037343616678841463726712390780448 \\ \alpha &= 0.4976413249969482421875 \end{aligned}$$

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

$$O\left(\left(2 - 5 \cdot 10^{-16}\right)^n\right).$$

4 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 2.5, 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^{-16}. Although Lemma 2.5 seems powerful, we lost a lot while applying it. In particular, the worst trade-off seems to happen in Section 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.

References

[1] Daniel Binkele-Raible, Ljiljana Brankovic, Marek Cygan, Henning Fernau, Joachim Kneis, Dieter Kratsch, Alexander Langer, Mathieu Liedloff, Marcin Pilipczuk, Peter Rossmanith, and Jakub Onufry Wojtaszczyk. Breaking the 2^n-barrier for irredundance: Two lines of attack. *J. Discrete Algorithms*, 9(3):214–230, 2011.

[2] Andreas Björklund. Determinant sums for undirected hamiltonicity. In *51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 173–182. IEEE Computer Society, 2010.

[3] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: fast subset convolution. In *39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 67–74, 2007.

[4] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

[5] Chandra Chekuri and Rajeev Motwani. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98(1-2):29–38, 1999.

[6] Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *CoRR*, abs/1103.0534, 2011.

[7] Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theor. Comput. Sci.*, 411(40-42):3701–3713, 2010.

[8] Marek Cygan, Marcin Pilipczuk, and Jakub Onufry Wojtaszczyk. Capacitated domination faster than $O(2^n)$. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory*, volume 6139 of *Lecture Notes in Computer Science*, pages 74–80. Springer, 2010.

[9] Fedor Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer, 2010.

[10] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):1–32, 2009.

[11] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[12] J. K. Lenstra and A.H.G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[13] Daniel Lokshtanov, Daniel Marx, and Saket Saurabh. Known Algorithms on Graphs of Bounded Treewidth are Probably Optimal. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 777–789, 2011.

[14] François Margot, Maurice Queyranne, and Yaoguang Wang. Decompositions, network flows, and a precedence constrained single-machine scheduling problem. *Operations Research*, 51(6):981–992, 2003.

[15] Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *45th Symposium on Foundations of Computer Science (FOCS)*, pages 248–255, 2004.

[16] Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1065–1075, 2010.

[17] Johan M. M. van Rooij, Jesper Nederlof, and Thomas C. van Dijk. Inclusion/exclusion meets measure and conquer. In *17th Annual European Symposium (ESA)*, volume 5757 of *Lecture Notes in Computer Science*, pages 554–565. Springer, 2009.

[18] Gerhard J. Woeginger. Space and time complexity of exact algorithms: Some open problems (invited talk). In Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors, *IWPEC*, volume 3162 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 2004.

[19] Gerhard J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008.