Scalable Neighborhood Local Search for Single-Machine Scheduling: Parameterized Complexity and Experiments (Full version)

Anonymous submission

Abstract

We study the problem of scheduling jobs on a single machine with sequence-dependent setup times minimizing the makespan. This notoriously NP-hard problem is highly relevant in practical productions. To deal with large instances in practice, it requires for heuristics that provide good solutions quickly. In this paper, we follow the approach of parameterized local search. That is, we aim to replace a given solution by a better solution having distance at most k in a pre-defined distance measure. This is done in a hill climbing manner, until a locally optimal solution is reached. We analyze the trade-off between k and the algorithm's running time for four natural distance measures in the solution space. For example: swapping k pairs of jobs in the sequence or rearranging k consecutive jobs. For two distance measures, we show that finding an improvement for given k can be done in FPT time for k, while such an FPT algorithm for the other two distance measures is unlikely. We experimentally evaluate the solution quality of our local search approaches. In particular, we evaluate the combination of different distance measures and analyze to which extent these provide good solutions within a reasonable running time even when additionally trying to minimize the total tardiness. Motivated by a practical use case, we consider setup times of a real-word injection molding machine.

Introduction

Finding orderings in which products are manufactured on a machine is among the most important problem families in combinatorial optimization and is highly relevant in practical production scenarios. It finds application in many real-world scenarios like the incider industry (Riahi et al. 2018) and solar cell industry (Chen et al. 2013). Since the mid-1960s, scheduling problems are extensively studied for a wide range of shop floor models, target functions, and setup information (Allahverdi 2015).

In this work, we study the task of scheduling jobs on a single machine with sequence dependent setup times under the goal of minimizing the makespan, that is, the completion time of the last job in the schedule. In this problem, one is given a set of jobs, each with an individual processing time, a deadline, and some identifier for a product type. Additionally, one is given a setup matrix specifying the additional time it takes when the machine is set up from one type to another. The goal is to find a sequence of all jobs such that every job meets its deadline and the completion time of the last job is minimal. Our work is motivated by the real-world production of the company X[1]. On a single injection molding machine, X manufactures products that come in eight different material/color combinations.

This problem is NP-hard as it generalizes the famous TRAVELING SALESPERSON PROBLEM (TSP). Due to the NP-hardness, instances with a large number of jobs are usually solved using heuristics. A common approach to tackle scheduling problems are genetic algorithms (OuYang and Xu 2014; Liao and Juan 2007). Since the applications of genetic algorithms usually have a large number of parameters that play an important role in their performance (Chiong and Dhakal 2009), it can be very difficult to reconstruct studied algorithms (Riahi, Newton, and Sattar 2021) and make them produce equally good results for a new use case. Moreover, genetic algorithms usually do not provide a (local) optimality guaranty for the returned solution.

A further important class of heuristics are local search algorithms, which usually have a simple design and provide a local optimality guaranty on the returned solution: In a hill climbing manner, one aims to improve a solution by performing *one* small change on a current solution, until no further improvement can be found. Variable Neighborhood Search (VNS) strategies that aim to find an improvement by swapping two jobs or inserting a job at another position in the schedule (Eddaly et al. 2009; Rego, Souza, and Arroyo 2012) are considered a state-of-the-art approach for single-machine scheduling with setup times (Allahverdi 2015).

We study another version of local search called *parameterized local search*. While in previous work, a solution was improved by *one* operation, the user may set a search radius k and extend the search space for possible improvements with up to k operations. This technique might prevent getting stuck in bad local optima. With this work, we aim to outline the trade-off between the running time of the heuristic and the size of the search radius k. Parameterized local search heuristics lately received much attention in the algorithmics community both from a theoretical (Bonnet et al. 2019; Dörnfelder et al. 2014; Gaspers et al. 2012; Guo et al. 2013; Komusiewicz et al. 2023; Szeider 2011) and practical (Gaspers et al. 2019; Grüttemeier, Komusiewicz, and Morawietz 2021; Hartung and Niedermeier 2013; Katzmann

[1]The company is anonymized for the review-process.

and Komusiewicz 2017) point of view. In case of sequencing problems, parameterized local search has been studied for classic TSP (Marx 2008) and for finding topological orderings of Bayesian networks (Grüttemeier, Komusiewicz, and Morawietz 2021). To the best of our knowledge, this is the first work studying parameterized local search for scheduling problems.

Our Contribution. The purpose of this work is to initiate the study of parameterized local search in context of single machine scheduling. We outline to which extent parameterized local search is a promising approach to handle our problem. We study four natural local search neighborhoods, analyze the parameterized complexity of the corresponding local search problems, and evaluate our findings experimentally for a large number of jobs and the real-world setup matrix from company Xs injection molding machine.

When considering the *insert neighborhood*, one aims to improve the solution by removing up to k jobs from the ordering and inserting them at new positions. When considering the *swap neighborhood*, one aims to improve the solution by consecutively swapping up to k pairs of jobs in a given schedule. Note that parameterized local search for these two neighborhoods is essentially an extension of existing work on classic local search (Eddaly et al. 2009; Rego, Souza, and Arroyo 2012). When considering the *window neighborhood*, one checks whether there exists a window consisting of k consecutive jobs in a given schedule, such that the solution can be improved by rearranging the jobs inside the window. When considering the *multi window neighborhood*, one can rearrange the jobs in multiple disjoint windows, of up to k consecutive jobs each, simultaneously to obtain an improved solution. Figure 1 shows one example of each studied neighborhood. Note that in the multi window distance, one is also able to consider the borders between windows. For example, given a schedule with product types $(1, 2, 3, 1)$ and $k = 2$, a schedule with product types $(2, 1, 1, 3)$ can be found with multi window local search by rearranging the windows $(1, 2)$ and $(3, 1)$ simultaneously. Moreover, the multi window neighborhood can be seen as an extension of the window neighborhood in the following sense: The set of possible schedules with multi window distance k is a superset of the schedules with window distance k from a given schedule. The multi window distance is closely related to a multi-inversions distance used for finding Bayesian network structures vial parameterized local search (Grüttemeier, Komusiewicz, and Morawietz 2021). Note that for all these neighborhoods, the problem of searching for a better schedule in the k-neighborhood becomes NP-hard since for sufficiently large k, the k-neighborhood contains all schedules.

In a theoretical part, we analyze the parameterized complexity of the four parameterized local search problems when parameterized by the search radius k. For the window neighborhood and the multi window neighborhood, we show that in time $k^{\min(k,t)} \cdot |I|^{\mathcal{O}(1)}$ one can find an improving schedule if one exists. Herein, $|I|$ denotes the total input size and t denotes the number of distinct product types in the instance. Note that in this running time, there is no exponential dependence on the total input size. Since the number of different product types manufactured on one machine is usually bounded by a small constant, the factor $k^{\min(k,t)}$ provides a good trade-off between the running time and the size of the search radius k. We complement this result by showing that for the insert and the swap neighborhood, there is probably no algorithm with running time $f(k) \cdot |I|^{\mathcal{O}(1)}$ for any computable function f.

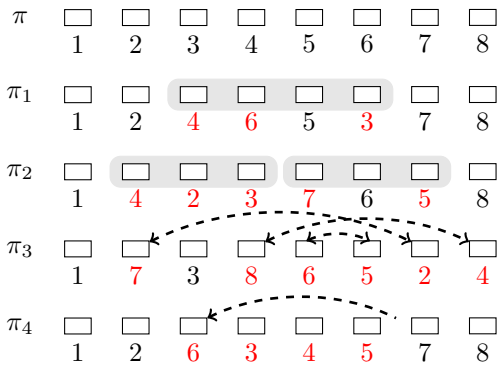

Figure 1: Example of the four considered distances to π. π_1 has window distance 4. π_2 has multi window distance 3. π_3 has swap distance 3. π_4 has insert distance 1.

Motivated by the theoretical results, we provide an experimental analysis of the algorithms applied on larger sets of jobs. While the theoretical results focus on minimizing the makespan when there exists a schedule without deadline violations, we extend the experimental analysis to instances where no feasible schedule exists and one aims to minimize the tardiness. Our experiments show that a VNS local search algorithm combining the 1-swap neighborhood and the window neighborhood for scalable k provides good results.

Full proofs of statements marked with (*) are provided in this supplementary material.

Preliminaries. We study a problem of sequencing jobs on a single machine with sequence-dependent family setup times. Formally, a *job* corresponds to a triple $j := (t_j, d_j, \tau_j)$. Herein, the value $t_j \in \mathbb{N}_0$ denotes the processing time of j, the value $d_j \in \mathbb{N}_0$ denotes the deadline of j, and τ_j is an identifier of a type of j. Let $\mathcal{T}$ denote the set containing all possible types. A *setup mapping* is a mapping $\ell : \mathcal{T} \times \mathcal{T} \to \mathbb{N}_0$. Given two types τ and τ', the value $\ell(\tau, \tau')$ is the time needed to set up the machine, if a job of type τ' is scheduled right after a job of type τ. Throughout this work, we assume that $\ell(\tau, \tau) = 0$ for every $\tau \in \mathcal{T}$ and that ℓ satisfies the triangle inequality. That is, $\ell(\tau_1, \tau_3) \leq \ell(\tau_1, \tau_2) + \ell(\tau_2, \tau_3)$ for every triple (τ_1, τ_2, τ_3) of types.

Let $J := \{j_1, \ldots, j_n\}$. A *schedule* is a permutation π of J. Given a schedule π of J, we define the *completion time of the ith job on* π as

$$C_\pi(i) := \begin{cases} t_{\pi(1)} & \text{if } i = 1, \\ C_\pi(i-1) + \ell(\tau_{\pi(i-1)}, \tau_{\pi(i)}) + t_{\pi(i)} & \text{if } i > 1. \end{cases}$$

The *makespan* $C^{\pi}_{\max}$ is then defined as the completion time of the last job on π. We may write $C_{\max}$ for the makespan if π is clear from the context. Furthermore, we define the *total tardiness of* π as

$$\sum_{i=1}^{n} \max(C_{\pi}(i) - d_{\pi(i)}, 0).$$

Throughout this work, we call a schedule π *feasible* if the total tardiness of π is 0. Intuitively, a schedule is feasible if every job meets its deadline. Motivated by our practical use case, we aim to find feasible schedules with minimum makespan.

MAKESPAN MINIMIZATION (MM)
Input: A set J of jobs and a setup mapping ℓ.
Task: Find a feasible schedule π of J that minimizes $C_{\max}$.

In the systematic *Graham notation* (Graham et al. 1979), MM is the problem $1 \mid \mathrm{ST}_{\mathrm{sd,f}} \mid C_{\max}$: Scheduling on one machine, when we have sequence-dependent family setup times and aiming to minimize the makespan $C_{\max}$.

Throughout this work, we let n denote the number of jobs in an instance (J, ℓ) of MM. Given two indices a, and b, we write $a <_{\pi} b$ if a precedes b in a sequence π. We let $\pi[a, b]$ denote the subsequence $(\pi(a), \pi(a+1), \ldots, \pi(b))$. Furthermore, we let $\pi \circ \sigma$ denote the concatenation of the sequences π and σ.

Parameterized Local Search. A *distance measure* $d : \pi \times \pi' \mapsto x \in \mathbb{N}$ maps a tuple of schedules to a non-negative integer. The distance measures considered in this work satisfy $d(\pi, \pi) = 0$ for every π and are symmetric, that is, $d(\pi, \pi') = d(\pi', \pi)$ for every π and π'.

In our heuristics for MM, we are given a schedule π and some integer k, and we aim to compute a better schedule that has distance at most k with π for some distance measure d. Formally, we solve the following computational problem.

d LS MM
Input: A set J of jobs, a setup mapping ℓ, a feasible schedule π of J, and an integer k.
Task: Find a schedule π' of J with $d(\pi, \pi') \leq k$ such that π' is *better* than π, that is, π' is feasible and $C^{\pi'}_{\max} < C^{\pi}_{\max}$.

Let d be a distance measure. We say that d LS MM is *fixed-parameter tractable (*FPT*) for* k if it can be solved in $f(k) \cdot n^{\mathcal{O}(1)}$ time for some function f. That is, the size n of the job set only contributes as a polynomial factor to the running time, while the exponential factor in the running time only depends on k. Thus, the running time of $f(k) \cdot n^{\mathcal{O}(1)}$ nicely outlines the trade-off between solution quality (radius size) and running time of the heuristic. If a problem is W[1]-hard when parameterized by k it is assumed that it is not fixed-parameter tractable for k. For a detailed introduction into parameterized complexity, we refer to the standard monograph (Cygan et al. 2015).

Local Search in Window Neighborhoods

We describe how to solve MM heuristically by algorithms based on parameterized local search, where we define neighborhoods in the solution space by rearranging k consecutive jobs.

In this section, we show that the corresponding local search problems for these two neighborhoods are both FPT when parameterized by k. The section is structured as follows. First, we formally introduce the distance measures leading to the local search problems WINDOW LS MM and MULTI WINDOW LS MM and we prove an observation on the solution structure of these problems, that we will later exploit in our algorithms. Second, we provide an efficient subroutine for improving windows of a schedule. Third, we describe how this subroutine can be used to obtain FPT algorithms for both local search problems.

Problem Definitions. We first consider the window distance. Let π and π' be permutations of n jobs. If $\pi = \pi'$, the schedules have window distance 0. Otherwise, the *window distance* is $b - a + 1$, where a (b) is the smallest (largest) index i of $[1, n]$ with $\pi(i) \neq \pi'(i)$. In the remainder of this work, we let WINDOW LS MM denote the local search problem d LS MM, where d is the window distance.

Given the schedules π and π' and the index set $[a, b]$ from the previous definition, we call the subsequence $\pi'[a, b]$ the *rearranged window*.

We next define the multi window distance. Let π and π' be schedules for the same job set J. We say that π and π' *decompose into* k*-intervals* for some integer k, if

$a)$ π and π' have length at most k, or
$b)$ there is some index $i \in [n - k + 1, n]$ such that $\{\pi(t) \mid t \in [i+1, n]\} = \{\pi'(t) \mid t \in [i+1, n]\}$, and the subsequences $\pi[1, i]$ and $\pi'[1, i]$ decompose into k-intervals.

Note that the decomposition into k-intervals is well-defined as $\{\pi(t) \mid t \in [i+1, n]\} = \{\pi'(t) \mid t \in [i+1, n]\}$ implies that the subsequences $\pi[1, i]$ and $\pi'[1, i]$ schedule the same jobs. The *multi window distance of* π *and* π' is then defined as the minimal value k for which π and π' decompose into k-intervals. In the remainder of this work, we let MULTI WINDOW LS MM denote the local search problem d LS MM, where d is the multi window distance.

Given two schedules π and π' that have multi window distance at most k, the indices i from the recursive definition above introduce a (not necessary unique) partition of $[1, n]$ into intervals $\{I_1, \ldots, I_m\}$. For one fixed partition, we call the subsequences $\pi'[I_i]$ the *rearranged windows*.

An Observation on the Solution Structure. Recall that the goal of this section is to provide efficient FPT algorithms for WINDOW LS MM and for MULTI WINDOW LS MM. This is done by adapting a dynamic programming algorithm that exploits an observation on the solution structure: We may limit our search space to solutions, where the jobs of each type are sorted by their deadlines.

A similar algorithm has previously been used by Cheng and Kovalyov 2001 to solve a special case of MM where all jobs of each type have the same processing time. Grüttemeier et al. 2023 also used this algorithmic idea to

solve MM with an unbounded number of processing times per type. However, the proof of the observation was omitted in their work. In this work, we provide the observation for the more general local search problems WINDOW LS MM and for MULTI WINDOW LS MM with an arbitrary number of different processing times per type.

To formally describe and prove the observation, we first introduce some terminology: Let (J, ℓ) be an instance of MM with types in $\mathcal{T}$. Furthermore, let $\tau \in \mathcal{T}$, and let π be a schedule for (J, ℓ). The *type-induced schedule* of π and τ is the subsequence $\pi[\tau]$ of π containing exactly the jobs of type τ. A schedule π is called *earliest due date schedule (EDDS)* if for every $\tau \in \mathcal{T}$, the type-induced schedule $\pi[\tau]$ is a schedule of all jobs of type τ in non-decreasing manner by their deadlines. A pair (a, b) of indices is called *type-inversion in* π, if $a <_\pi b$, the corresponding jobs j_a and j_b have the same type, and $d_{j_a} > d_{j_b}$. Note that π is an EDDS if and only if π has no type-inversions.

Proposition 1 (*)**.** *Let ℓ be a setup mapping such that $\ell(\tau, \tau) = 0$ for every type τ, and ℓ satisfies the triangle inequality. Moreover, let d be the distance measure for the window distance or the multi window distance.*

If (J, ℓ, π, k) is an instance of d LS MM *such that there exists an improving feasible schedule with distance at most k, then there is an improving feasible schedule π' with distance at most k, such that*

a) the rearranged window $\pi'[a, b]$ is an EDDS in case of window distance, and
b) all rearranged windows $\pi'[I_i]$ are EDDS in case of multi window distance.

Proof. We prove the proposition by first showing Statement b). Afterwards, we argue how the same arguments can be used to also show Statement a).

Let π' be a feasible schedule with multi window distance at most k from π such that $C^{\pi'}_{\max} < C^{\pi}_{\max}$. Furthermore, let ξ be the total number of type-inversions inside all rearranged windows $\pi'[I_i]$. Without loss of generality, we assume that π' is chosen in a way that ξ is minimal. We prove the statement by showing $\xi = 0$.

Assume towards a contradiction that $\xi > 0$. Then, there is one rearranged window $\pi'[I_i]$ containing a pair of jobs $j_a <_{\pi'[I_i]} j_b$, such that j_a and j_b have the same type $\tau\mathcal{T}$ and $d_{j_a} > d_{j_b}$. We change the job schedule inside $\pi'[I_i]$ in a way that the number of type-inversions in $\pi'[I_i]$ becomes strictly smaller, while not violating any deadline and not increasing $C^{\pi'}_{\max}$. Note that this does also not increase the multi window distance between π and π'. Therefore, such transformation contradicts the minimality of ξ.

Let σ be the subsequence of $\pi'[I_i]$ that contains all jobs between j_a and j_b on π. We consider the following cases.

Case 1: σ is the empty sequence**.** That is, no job is scheduled between j_a and j_b. We replace π' by π'_{new}, which we define as

$$\pi'_{\text{new}} := \pi'_{\text{pre}} \circ (j_b, j_a) \circ \pi'_{\text{suf}}.$$

Herein, π'_{pre}, π'_{suf} denote the (possibly empty) prefix and suffix of π' containing the jobs scheduled before and after (j_a, j_b). Intuitively, π'_{new} results from π' by swapping j_a and j_b. Since we only rearranged jobs inside $\pi[I_i]$, the resulting schedule π'_{new} has multi window distance at most k from π.

Note that (j_a, j_b) is not a type-inversion in π'_{new}, while for every other pair $(x, y) \neq (a, b)$ we have $x <_{\pi'_{\text{new}}} y$ if and only if $x <_{\pi'} y$. Consequently, the number of type-inversions in $\pi'_{\text{new}}[I_x]$ is smaller than in $\pi'[I_x]$. Moreover, since both jobs have the same type τ and $\ell(\tau, \tau) = 0$, swapping the jobs does not increase the sum of setup times and all jobs on π'_{pre} and π'_{suf} have the same completion time on π' and π'_{new}.

It remains to show that j_a and j_b meet their deadlines in π'_{new}. Let c_b denote the completion time of j_b under π'. By the fact that π' is feasible and (j_a, j_b) is a type-inversion in π', we have

$$c_b \underbrace{\leq}_{(1)} d_{j_b} \underbrace{<}_{(2)} d_{j_a}.$$

Since π'_{new} results from π' by swapping j_a and j_b, the new completion time of j_b is $c_b - t_{j_a} \leq c_b$. Thus, by (1), job j_b meets its deadline under π'_{new}. Furthermore, the new completion time of j_a is c_b, and therefore, j_a meets its deadline by (2).

Summarizing, swapping j_a and j_b inside $\pi'[I_i]$ results in a feasible schedule that has strictly less type-inversions in $\pi'[I_i]$ without increasing the makespan. This contradicts the minimality of ξ.

Case 2: σ is non-empty**.** We handle this case by transforming π' into a sequence π''_{new} satisfying the constraint of Case 1. Without loss of generality, we assume that no further job of type τ is scheduled between j_a and j_b on π'. Otherwise, we replace either j_a or j_b by a the corresponding job between. We define

$$\pi''_{\text{new}} := \pi'_{\text{pre}} \circ \sigma \circ (j_a, j_b) \circ \pi'_{\text{suf}}.$$

Intuitively, π''_{new} results from π' by interchanging j_a with the whole block σ. Note that we only rearranged jobs inside $\pi'[I_i]$ and therefore, π''_{new} has multi window distance at most k from π. Since the jobs on σ do not have type τ, we have $x <_{\pi''_{\text{new}}} y$ if and only if $x <_{\pi'} y$ for every pair of jobs (x, y) of the same type. Thus, the number of type-inversions in $\pi''_{\text{new}}[I_i]$ and $\pi'[I_i]$ are the same.

We next consider the makespan of π''_{new}. To this end, let τ_{pre} be the type of the last job on π'_{pre}. If the prefix is empty, we set $\tau_{\text{pre}} := \tau$. Furthermore, let τ^1_σ be the type of the first job on σ, and let τ^2_σ be the type of the last job on σ. Since π' and π''_{new} schedule the same jobs, the difference of the makespans can be expressed by the setup times that differ. Formally, this is

$$\begin{aligned}\Delta := C^{\pi''_{\text{new}}}_{\max} - C^{\pi'}_{\max} = &- \big(\ell(\tau_{\text{pre}}, \tau) + \ell(\tau, \tau^1_\sigma) + \ell(\tau^2_\sigma, \tau)\big) \\ &+ \big(\ell(\tau_{\text{pre}}, \tau^1_\sigma) + \ell(\tau^2_\sigma, \tau)\big).\end{aligned}$$

Since ℓ satisfies the triangle inequality, we have $\ell(\tau_{\text{pre}}, \tau) + \ell(\tau, \tau^1_\sigma) \geq \ell(\tau_{\text{pre}}, \tau^1_\sigma)$ and therefore $\Delta \leq 0$. Consequently, the makespan of π''_{new} is not greater than the makespan of π'.

It remains to show that each job meets its deadline in π''_{new}. Obviously, all jobs in π'_{pre} meet their deadlines, as they meet

their deadlines in π'. Next, observe that Δ is also the difference between the starting time step of job j_b in π''_{new} and in π'. Since $\Delta \leq 0$, the job j_b does not start at a later time step as it does in π'. Consequently, j_b and all jobs in π'_{suf} meet their deadlines. Since j_a is scheduled before j_b and we have $d_{j_a} > d_{j_b}$, job j_a also meets its deadline. Finally, since $\ell(\tau_{\text{pre}}, \tau) + \ell(\tau, \tau^1_\sigma) \geq \ell(\tau_{\text{pre}}, \tau^1_\sigma)$ by the triangle inequality, all jobs in σ do not start at a later time step in π''_{new} as they do in π'. Summarizing, all jobs meet their deadlines.

Note that π''_{new} satisfies the constraints of Case 1. Thus, by repeating the arguments from the previous case, we obtain a contradiction to the minimality of ξ. Since both cases are contradictory, Statement b) holds.

We next consider Statement a). Let π be a feasible schedule with window distance at most k from π such that $C^{\pi'}_{\max} < C^{\pi}_{\max}$. Furthermore, let ξ be the number of type-inversions in the rearranged window $\pi'[a, b]$, such that ξ is minimal among all such π'. Note that in the proof of b), rearranging one single rearranged window $\pi'[I_i]$ was sufficient to obtain a contradiction to the minimality of ξ in the case of $\xi > 0$. By repeating the same arguments for the single rearranged window $\pi'[a, b]$, we conclude $\xi = 0$. Thus, Statement a) holds. □

We would like to emphasize that the triangle inequality—which is needed for Proposition 1—is a natural probability of setup functions in practice: it appears to be unrealistic that setting up to an intermediate type τ' accelerates setting up a machine from a type τ_1 to another type τ_2. Even if that was the case, one would declare this intermediate setup as the standard setup from τ_1 to τ_2 and consider the updated setup function instead.

Subroutine Inside Windows. We now describe how to find the best EDDS inside a window. We later use this algorithm as a subroutine to solve WINDOW LS MM and MULTI WINDOW LS MM. To formally describe the subroutine, we introduce an auxiliary problem called INTERNAL MM. The intuition behind this problem is as follows: By Proposition 1 we can limit the search space inside one window to EDDS. Our window contains the jobs of some set J_1. A prefix of the schedule is already known to have some makespan θ. A suffix of the schedule is known to be some permutation σ of a job set J_2 disjoint from J_1. To keep track of the setup times when using an algorithm for INTERNAL MM as a subroutine, we keep track of the start type and the end type of the resulting EDDS.

In the remainder of this section, a schedule π is called θ-feasible for some integer θ, if $C^J_\pi(i) + \theta \leq d_{\pi(i)}$ for every i.

INTERNAL MM
Input: Two disjoint sets J_1 and J_2 of jobs, types τ_1, τ_2, a schedule σ of J_2, and some integer θ.
Task: Find an EDDS π of J_1 starting with a job of type τ_1 and ending with a job of type τ_2 such that $\pi \circ \sigma$ is a θ-feasible schedule for $J_1 \cup J_2$ and $\theta + C^{\pi\circ\sigma}_{\max}$ is minimal among all such π.

We next describe an algorithm solving INTERNAL MM. The algorithm is very closely related to algorithms for MM (Cheng and Kovalyov 2001; Grüttemeier, Komusiewicz, and Morawietz 2021). We provide it here for sake of completeness. In the following, $\mathcal{T}$ denotes the set of all types occurring in the set J_1.

Proposition 2 (*)**.** INTERNAL MM *can be solved in time*

$$\left(\prod_{\tau\in\mathcal{T}} (q_\tau + 1)\right) \cdot |I|^{\mathcal{O}(1)}.$$

Herein, $|I|$ is the total input size, and q_τ is the number of jobs with type τ in J_1.

Proof. Let $(J_1, J_2, \tau_1, \tau_2, \sigma, \theta)$ be an instance of INTERNAL MM. Furthermore, let $\mathcal{T} := \{1, \ldots, t\}$ denote the set of types occurring in J_1. We describe an algorithm based on dynamic programming. Given a type τ, we let EDD(τ) denote a sequence containing all jobs of type τ ordered in non-decreasing manner by their due-dates. Since we aim to find an EDDS π, we consider each EDD(τ) as a pre-ordered chain of jobs that keep their relative positions in π. The algorithm computes the best possible solution for prefixes of all EDD(τ) in a bottom-up manner, starting with the empty sequence, where all prefixes have length 0. To address partial solutions by their prefix-lengths, we use t-dimensional vectors $\vec{p}$ such that the τth entry of p corresponds to the length of the prefix of EDD(τ) that is scheduled in a corresponding partial solution. Furthermore, we let $\vec{p} - e_\tau$ denote the vector that is obtained from $\vec{p}$ when the τth entry is decreased by 1.

The entries of our dynamic programming table have the form $T[\vec{p}, \tau]$, where $\tau \in \mathcal{T}$, and $\vec{p}$ is a t-dimensional vector with $p_i \in [0, |\text{EDD}(i)|]$ for every $i \in \mathcal{T}$. Each entry stores the minimum value of $C^{\pi'}_{\max} + \theta$ over all θ-feasible schedules π' ending with a job of type τ and containing exactly the jobs indicated by the prefix-vector $\vec{p}$. If no such feasible schedule exists, the entry stores ∞.

We fill the table for increasing values of $|\vec{p}| := \sum_{i=1}^{t} p_i$. For $|\vec{p}| = 0$, that is, for $\vec{p} = \vec{0}$, we set $T[\vec{p}, \tau] = \theta$ for every τ. Next, for all $\vec{p}$ with $|\vec{p}| > 0$ and $p_\tau = 0$, we set $T[\vec{p}, \tau] = \infty$. The recurrence to compute an entry with $|\vec{p}| > 0$ and $p_\tau > 0$ is

$$T[\vec{p}, \tau] := \min_{\substack{\tau'\in\mathcal{T} \\ \text{feas}(\tau',\tau)}} T[\vec{p} - e_\tau, \tau'] + \ell(\tau', \tau) + t_{\text{EDD}(\tau)[p_\tau]}.$$

Herein, feas(τ', τ) is either true or false depending on whether the p_τth job on EDD(τ) can be scheduled after a job of type τ' without violating its deadline. This can be evaluated by checking if $T[\vec{p} - e_\tau, \tau'] + \ell(\tau', \tau) + t_{\text{EDD}(\tau)[p_\tau]} \leq d_{\text{EDD}(\tau)[p_\tau]}$.

After the table is filled, we consider $T[\vec{q}, \tau_2]$, where $\vec{q}$ is the t-dimensional vector, where the τth entry corresponds to the total number of jobs with type τ in J_1. We check, whether the schedule σ for J_2 is $T[\vec{q}, \tau_2] + \ell(\tau, \tau_{\sigma[1]})$-feasible. If this is the case, we compute the corresponding sequence π via traceback. Otherwise, we report that no solution for this instance exists.

We next consider the running time of the algorithm. The number of table entries is the product of $|\mathcal{T}|$ and the number of possible prefix vectors $\vec{p}$. Since the values of the τth

entry of such prefix vector range from 0 to q_τ, there are at most $\prod_{\tau \in \mathcal{T}}(q_\tau + 1)$ such prefix vectors. Since each entry can be computed in $|I|^{\mathcal{O}(1)}$ time, the algorithm has the stated running time. □

The factor $\prod_{\tau \in \mathcal{T}}(q_\tau + 1)$ from the running time relies on the fact that we limit our search space to EDDS. This limitation is done by merging sorted lists for each type. If we were not limiting our search space to EDDS, we can instead 'merge' lists of size one. Doing this, eventually leads to a running time of $2^{|J_1|} \cdot |I|^{\mathcal{O}(1)}$, which is as good as classic dynamic programming over subsets (Held and Karp 1962). Therefore, the algorithm behind Proposition 2 can be seen as an improvement over a $2^{|J_1|} \cdot |I|^{\mathcal{O}(1)}$-time algorithm by exploiting the solution structure of EDDS.

Recall that we want to use the algorithm behind Proposition 2 as a subroutine to solve the window local search problems in FPT time for k. To this end, note that the stated running time implies that INTERNAL MM can be solved in $k^t \cdot |I|^{\mathcal{O}(1)}$ time, where $k := |J_1|$ and t is the number of different types of jobs occurring in J_1. Note that $t \leq k$. This holds due to the fact that the product $\prod_{\tau \in \mathcal{T}}(q_\tau + 1)$ is maximal if all q_τ have roughly the same size $\frac{k}{t}$.

Fixed-Parameter Algorithms. We now describe how the algorithm behind Proposition 2 together with the observation from Proposition 1 can be used to obtain fixed-parameter tractability for the search radius k.

Theorem 1 (*). WINDOW LS MM *can be solved in* $k^{\min(k,t)} \cdot n^{\mathcal{O}(1)}$ *time, where* t *denotes the number of distinct types occurring in the input instance.*

Proof. The algorithm is straight forward: we iterate over every possible start position of a rearrangement window of length k, and use the algorithm behind Proposition 2 to check if rearranging this window leads to an improvement.

Formally, this is done as follows: For every possible $i \in [1, n-k]$ and every combination of two types τ' and τ'', we solve the INTERNAL MM-instance

$$(J_1, J_2, \tau', \tau'', \sigma, \theta),$$

where J_1 is the set of jobs in $\pi[i, i+k]$, σ is the (possibly empty) sequence of jobs scheduled after the $(i+k)$th job on π, and J_2 is the (possibly empty) set containing the jobs on σ. Moreover, we set $\theta := 0$, if $i = 1$ or $\theta := C_{\max}^{\pi[1,i-1]} + \ell(\tau_{\pi(i-1)}, \tau')$, if $i > 1$. If we found an improvement, we return the improving schedule. Otherwise, we report that no improvement is possible.

The correctness of the algorithm follows by the fact that we consider every combination of (i, τ', τ''), and that it is sufficient to find the best EDDS in a rearranged window due to Proposition 2 a). For the running time, note that each instance of INTERNAL MM can be solved in $k^{\min(k,t)} \cdot n^{\mathcal{O}(1)}$ time. Since we solve at most $n \cdot t^2$ such instances, the algorithm has the claimed running time. □

A similar result can be obtained for the multi window distance by using dynamic programming to find the decomposition into k-intervals.

Theorem 2 (*). MULTI WINDOW LS MM *can be solved in* $k^{\min(k,t)} \cdot n^{\mathcal{O}(1)}$ *time, where* t *denotes the number of distinct types occurring in the input instance.*

Proof. We show that MULTI WINDOW LS MM is FPT for the search radius k by providing an algorithm based on dynamic programming. Let (J, ℓ, π, k) be an instance of MULTI WINDOW LS MM.

Before we describe the algorithm, we provide some intuition. When aiming to find the best schedule that has multi window distance at most k from π we aim to find the best positions to cut π into windows. These windows must contain at most k elements. Thus, the last window contains the last $k' \leq k$ elements on π. Consequently, we can find a solution by finding the best cut position j for the last window and combine it with the best solution for $\pi[1, j-1]$. To find a solution of the last window, we use the algorithm behind Proposition 2. We use this recursive solution structure to define a recurrence to fill a dynamic programming table.

To formally describe the algorithm, we first introduce some notation. Given some $\pi[a, b]$, we let $J(\pi[a, b])$ denote the set of jobs scheduled in $\pi[a, b]$. Moreover, we let ε denote the empty sequence, and we let $\mathcal{L}(J_1, J_2, \tau_1, \tau_2, \sigma, \theta)$ denote the makespan of a solution of INTERNAL MM.

Our dynamic programming table T has entries of type $T[i, \tau]$ with $i \in [1, n]$ and $\tau \in \mathcal{T}$. Each entry stores the minimum makespan of a schedule of the jobs in $\pi[1, i]$ that ends with a job of type τ and has multi window distance at most k to $\pi[1, i]$. If no such schedule exists, the entry stores ∞.

The table is filled for increasing values of i. If $i \leq k$, we set

$$T[i, \tau] := \min_{\tau' \in \mathcal{T}} \mathcal{L}(J(\pi[1, i]), \emptyset, \tau', \tau, \varepsilon, 0).$$

Note that $T[i, \tau]$ corresponds to the makespan of an optimal EDDS of $\pi[1, i]$ ending with type τ. For entries with $i > k$, we set

$$T[i, \tau] := \min_{j \in [i-t, i-1]} \min_{\substack{\tau' \in \mathcal{T} \\ \tau'' \in \mathcal{T}}} \mathcal{L}(J(\pi[j+1, i]), \emptyset, \tau'', \tau, \varepsilon, \theta(j, \tau', \tau''))$$

with $\theta(j, \tau', \tau'') := T[j, \tau'] + \ell(\tau', \tau'')$.

The recurrence described above matches our intuition as follows: We find the best cut position for the last window by iterating over all possible j. We then combine an optimal schedule for this last window $\pi[j+1, i]$ with a partial solution of the schedule $\pi[1, j]$ by solving INTERNAL MM for an instance with $\theta(j, \tau', \tau'') = T[j, \tau'] + \ell(\tau', \tau'')$.

After the table is filled, we can compute the makespan of the solution by evaluating $\min_{\tau \in \mathcal{T}} T[n, \tau]$. The corresponding schedule can be found via traceback. The correctness of the algorithm follows by the fact that we consider every possible combination of (j, τ', τ''), and that it is sufficient to find the best EDDS in all rearranged windows due to Proposition 1.

We next analyze the running time of the algorithm. The dynamic programming table T has $n \cdot t$ entries; recall that t is the number of distinct types. To compute a single entry,

we iterate over each combination of j, τ', and τ''. Consequently, one table entry can be computed with $k \cdot t^2$ evaluations of the subroutine. Since INTERNAL MM can be solved in $k^{\min(k,t)} n^{\mathcal{O}(1)}$ time, the algorithm has the claimed running time. □

Swap distance and insert distance

In this section, we consider the local search problems SWAP LS MM and INSERT LS MM that ask for better schedules with small swap and insert distance, respectively. To this end, we first formally define these distance measures.

Two distinct schedules π and π' have *swap distance* one, if there are indices $a \in [1,n]$ and $b \in [a+1,n]$ such that $\pi(a) = \pi'(b)$, $\pi(b) = \pi'(a)$, and for each $i \in [1,n] \setminus \{a,b\}$, $\pi(i) = \pi'(i)$. Similarly, π and π' have *insert distance* one, if there are indices $a \in [1,n]$ and $b \in [1,n]$, such that after removing job $\pi(a)$ from π and inserting it at index b, one obtains π', that is, if $\pi' = \pi^-[1,b] \circ (\pi(a)) \circ \pi^-[b,n-1]$ for $\pi^- := \pi[1,a-1] \circ \pi[a+1,n]$.

Let $k > 1$. The schedules π and π' have swap (insert) distance at most k, if there is a schedule π'' such that π and π'' have swap (insert) distance one and π'' and π' have swap (insert) distance at most $k-1$. The swap (insert) distance of π and π' is then defined as the smallest value k for which π and π' have swap (insert) distance at most k.

In the following, we show that for the the swap and insert distance, FPT-algorithms for the search radius k are unlikely. More precisely, we show that even if each job has processing time one and the given schedule π tardiness zero, FPT-algorithms for k plus many other structural parameters are unlikely. On the positive side, we show that a simple brute force algorithm with running time n^{2k+1} is possible for both distance measures.

Theorem 3. SWAP LS MM *and* INSERT LS MM *are* W[1]*-hard when parameterized by*

- *the search radius k plus*
- *the largest setup time between any two types plus*
- *the largest finite deadline.*

This holds even on instances where

- *the initial schedule π has tardiness zero,*
- *each job has processing time one,*
- *each job has its unique type,*
- *the setup mapping assigns only four distinct values,*
- *the setup mapping fulfills the triangle inequality, and*
- *there is an optimal schedule within distance at most k of the initial schedule π.*

Proof. First, we show the statement for SWAP LS MM. The statement for INSERT LS MM then follows by the fact that two schedules π and π' have insert distance of at most $2k$ if they have swap distance at most k and in the SWAP LS MM-instance we construct, the initial solution is k-swap optimal if and only if it is globally optimal.

We present a parameterized reduction from MULTICOLORED CLIQUE which is W[1]-hard when parameterized by the size of the sought clique (Downey and Fellows 2013).

MULTICOLORED CLIQUE
Input: A graph $G = (V,E)$ and an integer k such that G is k-partite.
Question: Is there a clique of size k in G, that is, a set of k pairwise adjacent vertices?

Let $I := (G := (V,E), k)$ be an instance of MULTICOLORED CLIQUE with $k > 1$ and k-partition $(V_1, \dots, V_k)$. We describe how to obtain in polynomial time an equivalent instance $I' := (J, \ell, \pi, k')$ of SWAP LS MM fulfilling all stated restrictions.

Intuition and construction: The set J of jobs consists of two parts: A set J_V of $(k+1) \cdot |V|$ jobs that resemble the vertices of V and a set J_P of $2 \cdot k \cdot (k-1)$ auxiliary jobs. The initial schedule π starts with all jobs of J_P. The general idea of the reduction is that one can only improve over the initial schedule by swapping all jobs resembling vertices of a size-k clique in G with jobs of J_P.

Next, we start the formal definition of I'. For each vertex $v \in V$, we introduce a set of jobs J_v consisting of the following $k+1$ jobs: $\mathrm{start}^v, \mathrm{end}^v$, and for each $i \in [1,k-1]$ a job adj_i^v. Fix an arbitrary ordering of the vertices of V and let $V(i)$ denote the ith vertex of V according to this ordering. We set $\pi_V := \pi_{V(1)} \circ \cdots \circ \pi_{V(n)}$, where $\pi_v := (\mathrm{start}^v, \mathrm{adj}_1^v, \dots, \mathrm{adj}_{k-1}^v, \mathrm{end}^v)$ for each vertex $v \in V$.

Let $P := \{(i,j) \mid i \in [1,k], j \in [i+1,k]\}$ denote the set of ordered pairs of $[1,k] \times [1,k]$. We introduce for each pair $(i,j) \in P$ the set $J_{(i,j)}$ consisting of the four jobs: $\mathrm{start}^{(i,j)}, \mathrm{end}^{(i,j)}, \mathrm{aux}_{j-1}^i$, and aux_i^j. Let $P(i)$ denote the ith smallest pair of P with respect to the lexicographical order. We set $\pi_P := \pi_{P(1)} \circ \cdots \circ \pi_{P(|P|)}$, where $\pi_{(i,j)} := (\mathrm{start}^{(i,j)}, \mathrm{aux}_{j-1}^i, \mathrm{aux}_i^j, \mathrm{end}^{(i,j)})$ for each pair $(i,j) \in P$. Finally, we set $J := \bigcup_{(i,j) \in P} J_{(i,j)} \cup \bigcup_{v \in V} J_v$ and $\pi := \pi_P \circ \pi_V$, that is,

$$\pi := \underbrace{\pi_{P(1)} \circ \ldots \circ \pi_{P(|P|)}}_{\text{verification gadget}} \circ \underbrace{\pi_{V(1)} \circ \ldots \circ \pi_{V(|V|)}}_{\text{vertex gadget}}.$$

We define the processing time of each job as one and define the search radius $k' := k \cdot (k-1)$. The idea is that the search radius k' is large enough to ensure that for all $i \in [1,k]$, we are able to swap all jobs of $J_{v_i} \setminus \{\mathrm{start}^{v_i}, \mathrm{end}^{v_i}\}$ with auxiliary jobs. By defining the setup times properly, we can thus guarantee, that such a swap is improving if and only if there is a clique of size k in G.

It remains to define the deadline of each job and the setup mapping ℓ. As stated above, each job has its unique type. Hence, we may identify a job by its type. Let $\alpha := 4 \cdot |P| = 4\binom{k}{2} = 2k^2 - 2k$. For each $x \in [1,|P|]$, we set the deadline of $\mathrm{start}^{P(x)}$ to $1 + 4 \cdot (x-1) \cdot (\alpha+2)$ and the deadline of $\mathrm{end}^{P(x)}$ to $1 + (4x-1) \cdot (\alpha+2)$. For all other jobs of J, we set the deadline to ∞.

In the following, we define the setup mapping ℓ. Since each job has its unique type, we will describe the setup times between any two jobs. We define the setup times between any pair of consecutive jobs in π_V as α and the setup time from the last job of π_P (that is, $\mathrm{end}^{P(|P|)}$) to

the first job of π_V (that is, $\text{start}^{V(1)}$) as α. Similarly, for each pair $(i,j) \in P$, we define the setup times between any pair of consecutive jobs in $\pi_{(i,j)}$ as $\alpha+1$ and for each $x \in [1, |P|-1]$, we define the setup time from $\text{end}^{P(x)}$ to $\text{start}^{P(x+1)}$ as α.

Based on the so far defined setup times, we can derive the makespan of π.

Observation 1. *The makespan of π is exactly $|J| \cdot (\alpha+1) + 3 \cdot |P| - 1 < |J| \cdot (\alpha+1) + \alpha$.*

We now continue to describe the setup times between any pair of remaining jobs. For each $i \in [1,k]$ and each vertex $v \in V_i$, we define the setup times between any pair of consecutive jobs in $(\text{start}^v, \text{aux}^i_1, \ldots, \text{aux}^i_{k-1}, \text{end}^v)$ as α. Moreover, for each pair $(i,j) \in P$ and each vertex $v \in V_i$, we define the setup time from $\text{start}^{(i,j)}$ to adj^v_{j-1} as α. Similarly, for each pair $(i,j) \in P$ and each vertex $v \in V_j$, we define the setup time from adj^v_i to $\text{end}^{(i,j)}$ as α.

Let (i,j) be a pair of P, let u be a vertex of V_i, and let v be a vertex of V_j. If $\{u,v\}$ is an edge of E, we define the setup time from adj^u_{j-1} to adj^v_i as α. Otherwise, that is, if $\{u,v\}$ is not an edge of E, we define the setup time from adj^u_{j-1} to adj^v_i as 2α. Essentially, this resembles the validation gadget of the reduction. That is, if adj^v_i directly follows adj^u_{j-1} in any schedule π', then $\{u,v\}$ is an edge in E or the setup time from adj^u_{j-1} to adj^v_i is 2α. The latter then implies that π' is not a better schedule than π.

For any other pair (x,y) of distinct jobs of J, we define the setup time from x to y as 2α.

This completes the definition of the setup mapping ℓ. Note that each job of J finishes not later than its deadline in π. Hence, π has total tardiness zero.

Next, we summarize the main implications of the defined setup times. Let π' be a feasible schedule with fewer makespan than π. Since the makespan of π is $|J| \cdot (\alpha+1) + 3 \cdot |P| - 1 < |J| \cdot (\alpha+1) + \alpha$, the setup time between consecutive jobs in π' is less than 2α. Hence, we now describe for several jobs the possibilities which job may follow each other in π'.

Let (i,j) be a pair of P, let $v_i \in V_i$, and let $v_j \in V_j$.

- The job $\text{adj}^{v_i}_{j-1}$ may only directly follow $\text{start}^{(i,j)}$ or $\text{adj}^{v_i}_{j-2}$ (if $j=2$, $\text{adj}^{v_i}_{j-2} := \text{start}^{v_i}$).
- The job $\text{adj}^{v_j}_i$ may only be directly followed by $\text{end}^{(i,j)}$ or $\text{adj}^{v_j}_{i+1}$ (if $i = k-1$, $\text{adj}^{v_j}_{i+1} := \text{end}^{v_j}$).
- The job $\text{adj}^{v_j}_i$ may only directly follow $\text{adj}^{v_i}_{j-1}$ if $\{v_i, v_j\}$ is an edge of E.
- If $j > 2$, the job aux^i_{j-1} may only directly follow $\text{start}^{(i,j)}$ or adj^i_{j-2}.
- If $j = 2$, the job aux^i_{j-1} may only directly follow $\text{start}^{(i,j)}$ or start^w for some vertex $w \in V_i$.
- If $i < k-1$, the job aux^j_i may only be directly followed by $\text{end}^{(i,j)}$ or adj^j_{i+1}.
- If $j = k-1$, the job aux^j_i may only be directly followed by $\text{end}^{(i,j)}$ or end^w for some vertex $w \in V_j$.

Correctness: Next, we show the correctness of the reduction.

($\Rightarrow$) Let K be a clique of size k in G and let v_i denote the unique vertex of $K \cap V_i$ for each $i \in [1,k]$.

We show that there is a better schedule π' for J which has swap distance at most k' with π. More precisely, we show that there is a feasible schedule π' with makespan exactly $|J| \cdot (\alpha+1) - \alpha$ which has swap distance at most k' with π. Note that a feasible schedule with makespan $|J| \cdot (\alpha+1) - \alpha$ is optimal, since the setup time between any two jobs is at least α and each job has a processing time of one.

We obtain π' by performing the following k' swaps: for each $i \in [1,k]$ and each $j \in [1, k-1]$, we swap $\text{adj}^{v_i}_j$ with aux^i_j. That is, for each pair $(i,j) \in P$, the partial schedule $\pi'_{(i,j)}$ between $\text{start}^{(i,j)}$ and $\text{end}^{(i,j)}$ is exactly $\pi'_{(i,j)} := (\text{start}^{(i,j)}, \text{adj}^{v_i}_{j-1}, \text{adj}^{v_j}_i, \text{end}^{(i,j)})$ and for each $i \in [1,k]$, the partial schedule π'_{v_i} between start^{v_i} and end^{v_i} is exactly $\pi'_{v_i} := (\text{start}^{v_i}, \text{aux}^i_1, \ldots, \text{aux}^i_{k-1}, \text{end}^{v_i})$. Moreover, for each vertex $w \in V \setminus K$, the partial schedule π'_w between start^w and end^w remains unchanged, that is, $\pi'_w := \pi_w$. The complete schedule π' is then $\pi' = \pi'_P \circ \pi'_V$, where $\pi'_P := \pi'_{P(1)} \circ \cdots \circ \pi'_{P(n)}$ and $\pi'_V := \pi'_{V(1)} \circ \cdots \circ \pi'_{V(n)}$.

We now show that the setup time between any two consecutive jobs in π' is α and that π' has total tardiness zero. Since the setup time between any two jobs is at least α, this then implies that π' is an optimal schedule of J.

First, we show that for each pair $(i,j) \in P$, the setup time between any two consecutive jobs in $\pi'_{(i,j)}$ is α. Let (i,j) be a pair of P. By construction, the setup time from $\text{start}^{(i,j)}$ to $\text{adj}^{v_i}_{j-1}$ is α since v_i is a vertex of V_i. Similarly, the setup time from $\text{adj}^{v_j}_i$ to $\text{end}^{(i,j)}$ is α since v_j is a vertex of V_j. Moreover, since K is a clique in G, $\{v_i, v_j\}$ is an edge in G. Hence, by construction, the setup time from $\text{adj}^{v_i}_{j-1}$ to $\text{adj}^{v_j}_i$ is α as well. Consequently, for each pair $(i,j) \in P$, the setup time between each pair of consecutive jobs of $\pi'_{(i,j)}$ is α. Moreover, for each $x \in [1, |P|-1]$, the setup time from $\text{end}^{P(x)}$ to $\text{start}^{P(x+1)}$ is defined as α. Consequently, the setup time between any two consecutive jobs in π'_P is $alpha$.

Since the setup time from the last job of π'_P (that is, $\text{end}^{P(|P|)}$) to the first job of π'_V (that is, $\text{start}^{V(1)}$) is defined as α, it remains to show that the setup time between any two consecutive jobs in π'_V is also α. By construction, for each vertex $v \in V \setminus K$, the setup time between any two consecutive jobs of $\pi_v = \pi'_v$ is α. Moreover, for each $x \in [1, n-1]$, the setup time from the last job of $\pi'_{V(x)}$ (that is, $\text{end}^{V(x)}$) to the first job of $\pi'_{V(x+1)}$ (that is, $\text{start}^{V(x+1)}$) is defined as α. The remaining setup times to consider are the setup times between consecutive jobs in π'_v for each vertex $v \in K$. Let $i \in [1,k]$. We show that the setup times between consecutive jobs in π'_{v_i} are also α. Since v_i is a vertex of V_i, by construction, the setup time between any two consecutive jobs of $\pi'_{v_i} = (\text{start}^{v_i}, \text{aux}^i_1, \ldots, \text{aux}^i_{k-1}, \text{end}^{v_i})$ is α.

Hence, the setup time between any two consecutive jobs

in π' is α. This implies that the makespan of π' is $|J| \cdot (\alpha + 1) - \alpha$, which is optimal, since each job has processing-time one and the setup time between any two jobs is at least α. Next, we show that π' has tardiness zero. Moreover, since for each pair $(i,j) \in P$, the job $\mathrm{start}^{(i,j)}$ has the same index in both schedules π and π', the job $\mathrm{start}^{(i,j)}$ finishes in π' not later than in π. The same holds for the job $\mathrm{end}^{(i,j)}$. Since π has total tardiness zero, this implies that π' also has tardiness zero. Consequently, π' is an optimal schedule for J which improves over π and has swap distance at most k' with π.

($\Leftarrow$) Let π' be a feasible schedule of J that improves over π and that has the smallest swap distance to π among all schedule of J that improves over π. We show that there is a clique of size k in G.

By the first direction, this then implies that π' is an optimal schedule of J and has swap distance at most k' with π.

Since the makespan of π is $|J| \cdot (\alpha + 1) + 4 \cdot |P| - 1 < |J| \cdot (\alpha + 1) + \alpha$ and each setup time between any two jobs of J is at least $\alpha = 4 \cdot |P|$ we obtain the following.

Observation 2. *The setup time between any two consecutive jobs in π' is less than 2α.*

Claim 1. *Let $x \in [1, |P|]$ and let $(i,j) = P(x)$. The job $\mathrm{start}^{(i,j)}$ has the same index in both π and π', that is, index $4 \cdot x - 3$. The job $\mathrm{end}^{(i,j)}$ has the same index in both π and π', that is, index $4 \cdot x$.*

Proof. We show this statement by induction over x. To this end, we first show the following statements:

1. For the base case, we show that $\mathrm{start}^{P(1)}$ is the first job of π'.
2. For each $x \in [1, |P|]$, we show that a) if $\mathrm{start}^{P(x)}$ has index $4 \cdot x - 3$ in π' and b) if for each $y \in [1, x-1]$, $\mathrm{start}^{P(y)}$ has index $4 \cdot y - 3$ in π' and $\mathrm{end}^{P(y)}$ has index $4 \cdot y$ in π', then $\mathrm{end}^{P(x)}$ has index $4 \cdot x$ in π'.
3. For each $x \in [2, |P|]$, we show that if $\mathrm{start}^{P(1)}$ is the first job of π' and if $\mathrm{end}^{P(x-1)}$ has index $4 \cdot (x-1)$ in π', then $\mathrm{start}^{P(x)}$ has index $4 \cdot x - 3$ in π'.

With these statements at hand, one can then show via induction over x that the statement of Claim 1 holds: The base case $x = 1$ of the induction follows by Statement 1 and Statement 2. For the inductive step, we assume that the statement of Claim 1 holds for all $y \in [1, x-1]$. The induction hypothesis then ensures that the conditions of Statement 2 and Statement 3 hold. Hence, both these statements then imply that the statement of Claim 1 holds for x.

First, we show Statement 1, that is, that $\mathrm{start}^{P(1)}$ is the first job of π'. This holds, since the deadline of $\mathrm{start}^{P(1)}$ is 1, each job of J has a processing-time of 1, and π' has total tardiness zero.

Next, we show Statement 2. Let $x \in [1, |P|]$ such that a) $\mathrm{start}^{P(x)}$ has index $4 \cdot x - 3$ in π' and b) for each $y \in [1, x-1]$, $\mathrm{start}^{P(y)}$ has index $4 \cdot y - 3$ in π' and $\mathrm{end}^{P(y)}$ has index $4 \cdot y$ in π'. Let i^* denote the index of $\mathrm{end}^{P(x)}$ in π'. We show that $i^* = 4 \cdot x$. Since the deadline of $\mathrm{end}^{P(x)}$ is less than $4 \cdot x \cdot (\alpha + 1)$, i^* is at most $4 \cdot x$. Hence, it remains to show that i^* is at least $4 \cdot x$. To this end, we first show that i^* is at least $4 \cdot x - 2$. Recall that for each $y \in [1, x-1]$, $\mathrm{start}^{P(y)}$ has index $4 \cdot y - 3$ in π' and $\mathrm{end}^{P(y)}$ has index $4 \cdot y$ in π'. Hence, for each $y \in [1, x-1]$, $i^* \neq 4 \cdot y - 2$, since the setup time from $\mathrm{start}^{P(y)}$ to $\mathrm{end}^{P(x)}$ is 2α. Similarly, for each $y \in [1, x-1]$, $i^* \neq 4 \cdot y - 1$, since the setup time from $\mathrm{end}^{P(x)}$ to $\mathrm{end}^{P(y)}$ is 2α. Consequently, i^* is at least $4 \cdot x - 2$. It remains to show that $i^* \neq 4 \cdot x - 2$ and $i^* \neq 4 \cdot x - 1$.

Since the setup time from $\mathrm{start}^{P(x)}$ to $\mathrm{end}^{P(x)}$ is defined as 2α, $i^* \neq 4 \cdot x - 2$. Assume towards a contradiction that $i^* = 4 \cdot x - 1$ and let $\widehat{\mathrm{job}}$ be the job with index $i^* - 1 = 4 \cdot x - 2$ in π'. We show that the setup time from $\mathrm{start}^{P(x)}$ to $\widehat{\mathrm{job}}$ is 2α or the setup time from $\widehat{\mathrm{job}}$ to $\mathrm{end}^{P(x)}$ is 2α. Let $(i,j) = P(x)$. If the setup time from $\mathrm{start}^{P(x)}$ to $\widehat{\mathrm{job}}$ is less than 2α, then by construction, $\widehat{\mathrm{job}}$ is either the job aux^i_{j-1} or the job adj^v_{j-1} for some vertex $v \in V_i$. If the setup time from $\widehat{\mathrm{job}}$ to $\mathrm{end}^{P(x)}$ is less than 2α, then by construction, $\widehat{\mathrm{job}}$ is either the job aux^j_i or the job adj^v_i for some vertex $v \in V_j$. Since $i \neq j$, not both of these conditions hold. This implies that at least one of the setup times incident with $\widehat{\mathrm{job}}$ is 2α which leads to a contradiction due to Observation 2. Consequently, the index of $\mathrm{end}^{P(x)}$ in π' is exactly $4 \cdot x$. This completes the proof of Statement 2.

Finally, we show Statement 3. Let $x \in [2, |P|]$ such that $\mathrm{start}^{P(1)}$ is the first job of π' and $\mathrm{end}^{P(x-1)}$ has index $4 \cdot (x-1)$ in π'. We show that, $\mathrm{start}^{P(x)}$ has index $4 \cdot x - 3$ in π'. To this end, recall that the job $\mathrm{end}^{P(x-1)}$ is the unique job $\widehat{\mathrm{job}}$ of J such that the the setup time from $\widehat{\mathrm{job}}$ to $\mathrm{start}^{P(x)}$ is less than 2α. Hence, Observation 2 implies that $\mathrm{start}^{P(x)}$ is either the first job of π' or $\mathrm{start}^{P(x)}$ is the successor of $\mathrm{end}^{P(x-1)}$ in π'. By assumption, $\mathrm{start}^{P(1)}$ is the first job of π', we conclude that $\mathrm{start}^{P(x)}$ is the successor of $\mathrm{end}^{P(x-1)}$ in π'. Since $\mathrm{end}^{P(x-1)}$ has index $4 \cdot (x-1)$ in π', $\mathrm{start}^{P(x)}$ has index $4 \cdot x - 3$ in π'. ■

Due to this property, we call the jobs of $J_F := \{\mathrm{start}^{(i,j)}, \mathrm{end}^{(i,j)} \mid (i,j) \in P\}$ *fixed*. Moreover, we call a job of J *early*, if that job has index at most $4 \cdot |P|$ in π'. Consequently, for each early job $\widehat{\mathrm{job}}$ of $J \setminus J_F$, there is some $x \in [1, |P|]$, such that $\widehat{\mathrm{job}}$ directly follows $\mathrm{start}^{P(x)}$ in π', or $\mathrm{end}^{P(x)}$ directly follows $\widehat{\mathrm{job}}$ in π'. By construction, this implies that for each vertex $v \in V$, both jobs start^v and end^v are not early, since the setup time from each of these two jobs to some job of J_F is 2α and the setup time from each job of $J_F \setminus \{\mathrm{end}^{P(|P|)}\}$ to each of these two jobs is 2α.

Similarly, this also implies that for each other early job $\widehat{\mathrm{job}} \in J$, there is exactly one possible index for $\widehat{\mathrm{job}}$ in π'. Formally, let $i \in [1, k]$, let $j \in [1, k-1]$, and let v be a vertex of V_i. We distinguish two cases.

Case 1: $i \leq j$. By construction, the setup time

from $\mathrm{start}^{(i,j+1)}$ to adj_j^v (aux_j^i) is less than 2α and the setup time between any job of $J_F \setminus \{\mathrm{start}^{(i,j+1)}\}$ and adj_j^v (aux_j^i) is 2α. In other words, if adj_j^v (aux_j^i) is an early job, then adj_j^v (aux_j^i) directly follows $\mathrm{start}^{(i,j)}$ in π'.

Case 2: $i > j$**.** By construction, the setup time from adj_j^v (aux_j^i) to $\mathrm{end}^{(j,i)}$ to is less than 2α and the setup time between adj_j^v (aux_j^i) and each job of $J_F \setminus \{\mathrm{end}^{(j,i)}\}$ is exactly 2α. In other words, if adj_j^v (aux_j^i) is an early job, then $\mathrm{end}^{(j,i)}$ directly follows adj_j^v (aux_j^i) in π'.

For each such job $\widehat{\mathrm{job}}$, we call this unique index of $[1, 4 \cdot |P|]$ the *early-index* of $\widehat{\mathrm{job}}$. Note that for each job $\widehat{\mathrm{job}} \in \bigcup_{(i,j)\in P} J_{(i,j)}$, the early-index of $\widehat{\mathrm{job}}$ is equal to the index of $\widehat{\mathrm{job}}$ in π.

In the following, we show that for each $i \in [1, k]$, there is some vertex $v_i \in V_i$, such that some job of J_{v_i} is early. We first how the statement for $i = 1$.

Claim 2. *There is a vertex $v_1 \in V_1$ such that the job $\mathrm{adj}_1^{v_1}$ is early.*

Proof. Recall that the only possible way to improve over π is to reduce the number of setup times of $\alpha+1$ and that all these setup times occur within the first $4 \cdot |P|$ jobs of π. Since π' improves over π, there is some $i \in [1, k]$ and some $j \in [1, k-1]$ such that aux_j^i is not an early job. Let (i, j) be the lexicographically smallest pair such that aux_j^i is not an early job. Since aux_j^i is not an early job, some other job of J has the early-index of aux_j^i in π'. As shown above, the only other jobs that shares the early-index of aux_j^i are the jobs $\mathrm{adj}_j^{v_i}$ for each vertex $v_i \in V_i$. Let v_i be the unique vertex of V_i, such that the job $\mathrm{adj}_j^{v_i}$ is early.

We show that $i = j = 1$. Assume towards a contradiction that this is not the case. We consider two cases.

Case 1: $i > j$**.** Hence, in π, aux_j^i directly follows aux_{i-1}^j. Since the setup time between any two consecutive jobs in π' is less than 2α, and the setup time from aux_{i-1}^j to $\mathrm{adj}_j^{v_i}$ is 2α, aux_{i-1}^j is not an early job. Since $i > j$, $(j, i-1)$ is lexicographically smaller than (i, j), a contradiction.

Case 2: $i \leq j$**.** Recall that we assume that $i > 1$ or $j > 1$. This then implies that $j > 1$. Since aux_j^i is not an early job, aux_j^i does not directly follow $\mathrm{start}^{(i,j+1)}$ in π'. By construction, aux_{j-1}^i is the only other job of J from which the setup time to aux_j^i is less than 2α. Hence, aux_j^i directly follows aux_{j-1}^i in π' which implies that aux_{j-1}^i is not an early job. Thus, we encounter a contradiction since $(i, j-1)$ is lexicographically smaller than (i, j).

Consequently, aux_1^1 is not an early job, which implies that there is a vertex $v_1 \in V_1$ such that $\mathrm{adj}_1^{v_1}$ is an early job. ■

Next, we show that Claim 2 can be used as the base case to show that for each $i \in [1, k]$, there is some vertex $v_i \in V_i$, such that each job of J_{v_i} is early.

Claim 3. *Let $i \in [1, k]$, let $j \in [1, k-1]$, and let v_i be a vertex of V_i such that the job $\mathrm{adj}_j^{v_i}$ is early.*

1. *If $i \leq j+1$ and $j \leq k-2$, then the job $\mathrm{adj}_{j+1}^{v_i}$ is early.*
2. *If $i > j$ and $j > 1$, then the job $\mathrm{adj}_{j-1}^{v_i}$ is early.*
3. *If $i = j$, then there is some vertex $v_{i+1} \in V_{i+1}$ such that the job $\mathrm{adj}_i^{v_{i+1}}$ is early.*

Proof. First, we show the first statement. Let $i \leq j+1$ and let $j \leq k-2$. We show that the job $\mathrm{adj}_{j+1}^{v_i}$ is early. Since $i \leq j+1$, by construction, $\mathrm{adj}_j^{v_i}$ and $\mathrm{start}^{(i,j+2)}$ are the only two jobs of J from which the setup time to $\mathrm{adj}_{j+1}^{v_i}$ is less than 2α. By assumption, $\mathrm{adj}_j^{v_i}$ is an early job. Moreover, due to Claim 1, $\mathrm{start}^{(i,j+2)}$ is an early job. Consequently, $\mathrm{adj}_{j+1}^{v_i}$ is an early job, since the setup time between any two consecutive jobs in π' is less than 2α.

Next, we show the second statement. Let $i > j$ and $j > 1$. We show that the job $\mathrm{adj}_{j-1}^{v_i}$ is early. This statement follows by similar arguments as the first statement. Since $i > j$, by construction, the setup time from $\mathrm{adj}_{j-1}^{v_i}$ to any job of $J \setminus \{\mathrm{end}^{(j,i)}, \mathrm{adj}_j^{v_i}\}$ is defined as 2α. Recall that $\mathrm{end}^{V(n)}$ is the last job of π', since the setup time from $\mathrm{end}^{V(n)}$ to any other job of J is defined as 2α. Hence, some job of J directly follows $\mathrm{adj}_{j-1}^{v_i}$ in π'. By assumption, $\mathrm{adj}_j^{v_i}$ is an early job. Moreover, due to Claim 1, $\mathrm{end}^{(j,i)}$ is an early job. Consequently, $\mathrm{adj}_{j-1}^{v_i}$ is an early job, since the setup time between any two consecutive jobs in π' is less than 2α.

Finally, we show the third statement. Let $i = j$. We show that there is some vertex $v_{i+1} \in V_{i+1}$ such that the job $\mathrm{adj}_i^{v_{i+1}}$ is early. Since $\mathrm{adj}_i^{v_i}$ is early, $\mathrm{adj}_i^{v_i}$ directly follows $\mathrm{start}^{(i,i+1)}$ in π'. We consider the job $\widehat{\mathrm{job}}$ of J that directly follows $\mathrm{adj}_i^{v_i}$ in π'. Due to Claim 1, $\mathrm{end}^{(i,i+1)}$ directly follows $\widehat{\mathrm{job}}$ in π'. Since the setup time between any two consecutive jobs in π' is less than 2α, this implies that $\widehat{\mathrm{job}}$ is either the job aux_i^{i+1} or the job $\mathrm{adj}_i^{v_{i+1}}$ for some vertex $v_{i+1} \in V_{i+1}$. Recall that by construction, the setup time from $\mathrm{adj}_i^{v_i}$ to aux_i^{i+1} is 2α. Hence, for some vertex $v_{i+1} \in V_{i+1}$, the job $\mathrm{adj}_i^{v_{i+1}}$ is early. ■

With the base case of Claim 2 we can now show the following inductively due to Claim 3.

Claim 4. *For each $i \in [1, k]$, there is a vertex $v_i \in V_i$ such that each job of $J_{v_i} \setminus \{\mathrm{start}^{v_i}, \mathrm{end}^{v_i}\}$ is early.*

Proof. First, we show that for each $i \in [2, k]$, there is a vertex $v_i \in V_i$ such that $\mathrm{adj}_{i-1}^{v_i}$ is early. We show this statement by induction over i.

We consider the base case of $i = 2$. Due to 2, there is some vertex $v_1 \in V_1$ such that $\mathrm{adj}_1^{v_1}$ is early. Hence, by Statement 3, there is some vertex $v_2 \in V_2$ such that $\mathrm{adj}_1^{v_2}$ is early. For the inductive step assume that $i \in [3, k]$ and that there is a vertex $v_{i-1} \in V_{i-1}$ such that $\mathrm{adj}_{i-2}^{v_{i-1}}$ is early. Due to Statement 1, this implies that $\mathrm{adj}_{i-1}^{v_{i-1}}$ is early. Hence, by Statement 3, there is some vertex $v_i \in V_i$ such that $\mathrm{adj}_{i-1}^{v_i}$ is early.

Consequently, for each $i \in [2, k]$, there is a vertex $v_i \in V_i$ such that $\mathrm{adj}_{i-1}^{v_i}$ is early. Moreover, this proof also shows that for each $i \in [1, k-1]$, $\mathrm{adj}_i^{v_i}$ is early. Let $i \in [1, k]$. It remains to show that each job of $J_{v_i} \setminus \{\mathrm{start}^{v_i}, \mathrm{end}^{v_i}\}$

is early. This can be done in two steps. First, if $i \leq k-1$, one can show that $\mathrm{adj}_j^{v_i}$ is early for each $j \in [i, k-1]$. Due to Statement 1 this can be shown inductively since the base case for $j = i$ holds, that is, since $\mathrm{adj}_i^{v_i}$ is early. Second, if $i > 1$, one can show that $\mathrm{adj}_j^{v_i}$ is early for each $j \in [1, i-1]$. Due to Statement 2 this can be shown inductively since the base case for $j = i-1$ holds, that is, since $\mathrm{adj}_{i-1}^{v_i}$ is early. Consequently, for each $i \in [1, k]$, there is a vertex $v_i \in V_i$, such that each job of $J_{v_i} \setminus \{\mathrm{start}^{v_i}, \mathrm{end}^{v_i}\}$ is early. ■

For each $i \in [1, k]$, let v_i be a vertex of V_i such that each job of $J_{v_i} \setminus \{\mathrm{start}^{v_i}, \mathrm{end}^{v_i}\}$ is early. Since there are exactly $4 \cdot |P| = 4 \cdot \binom{k}{2}$ early jobs and $J_{v_i} \setminus \{\mathrm{start}^{v_i}, \mathrm{end}^{v_i}\}$ has size $k-1$, the set of all early jobs is exactly $J_F \cup \bigcup_{i \in [1,k]} J_{v_i} \setminus \{\mathrm{start}^{v_i}, \mathrm{end}^{v_i}\}$. In other words, for each $i \in [1, k]$, v_i is the unique vertex of V_i for which a job of J_{v_i} is early. It remains to show that $K := \{v_i \mid 1 \leq i \leq k\}$ is a clique in G.

Recall that the setup time between any two consecutive jobs of π' is less than 2α. Since for each pair $(i, j) \in P$, the job $\mathrm{adj}_i^{v_j}$ directly follows the job $\mathrm{adj}_{j-1}^{v_i}$ in π', this then implies by construction, that $\{v_i, v_j\}$ is an edge in G. Consequently, K is a clique in G.

This now implies that π is k'-swap optimal if and only if π is an optimal schedule.

To conclude the proof for SWAP LS MM, we now show that the constructed instance fulfills all described restrictions. We show that the W[1]-hardness of MULTICOLORED CLIQUE when parameterized by k transfers to SWAP LS MM when parameterized by the sum of stated parameters. To this end, we show that the sum of these parameters does not exceed $\mathcal{O}(k^4)$. By construction, $k' = k \cdot (k-1) \in \mathcal{O}(k^4)$ and the largest setup time between any two jobs is $2\alpha = 4k^2 - 4k \in \mathcal{O}(k^4)$. Moreover, $\mathrm{end}^{P(|P|)}$ has the largest finite deadline of all jobs, namely $1 + (4 \cdot |P| - 1) \cdot (\alpha + 2) = 1 + (2k^2 - 2k - 1) \cdot (2k^2 - 2k + 2) \in \mathcal{O}(k^4)$. Hence, the sum of all three parameters does not exceed $\mathcal{O}(k^4)$.

To show that I' fulfills all describes restrictions, we only have to show that the setup mapping ℓ fulfills the triangle inequality, since all other restrictions were discussed previously. Note that the triangle inequality holds since the setup time between any two jobs is at least α and at most 2α. Hence, a detour through any other sequence of jobs has total setup time at least 2α as well.

Finally, we show that the hardness result also transfers to INSERT LS MM. This follows from the fact that each schedule of swap distance at most k' to π has insert distance at most $2 \cdot k'$ to π. Since π is k'-swap optimal if and only if π is an optimal schedule, this implies that π is $(2 \cdot k')$-insert optimal if and only if π is an optimal schedule. Hence, $I'' := (J, \ell, \pi, k'')$ with $k'' := 2 \cdot k'$ is a yes-instance of INSERT LS MM if and only if I' is a yes-instance of SWAP LS MM and all hardness results transfer to INSERT LS MM as well. □

In other words, even on very restricted instances and for each computable function f, a running time of $f(k) \cdot |I|^{\mathcal{O}(1)}$ for SWAP LS MM or INSERT LS MM is unlikely. Still, a simple brute-force algorithm yields the following running time for both problems.

Algorithm 1 Pseudo code of Win+Swap

```
1:  Input Instance (J, ℓ) of MM and an initial schedule π
2:  k ← 4
3:  while time is not exceeded do
4:      if π is not 1-swap optimal then
5:          π ← better schedule with swap distance one
6:          k ← 4
7:          continue
8:      if π is k-window optimal then
9:          k ← k + 1
10:     else
11:         π ← better schedule with window distance ≤ k
12:         k ← 4
    return π
```

Theorem 4. SWAP LS MM *and* INSERT LS MM *can be solved in* n^{2k+1} *time.*

Essentially this follows by a simple brute-force algorithm that evaluates for all k consecutive applied swaps (inserts) in the solution quality of the obtained schedule. The latter can be done in $\mathcal{O}(n)$ time and enumerating all k consecutive applied swaps can be done in n^{2k} time.

Derived hill climbing algorithms

Based on the theoretical results of the previous sections, we derived four hill climbing local search algorithms: Win, Win+Swap, MW, MW+Swap. The pseudo code for Win+Swap is depicted in Algorithm 1.

The general idea of these hill climbing algorithm is to start with some initial solution π and $k = 4$. The algorithm then tries to replace the current solution by a better one with distance at most k. To evaluate whether there is a better solution with distance at most k, we use the algorithm behind Theorems 1 and 2. If no such better solution exists, the algorithm is stuck in a locally optimal solution. Aiming to escape this locally optimal solution, the value of k is incremented, which yields a larger neighborhood to consider. Otherwise, if there is a better schedule π' with distance at most k, we replace π by π' and set k back to 4. We always set k back to 4 after finding any better solution, because searching for a better solution with distance at most k can be performed faster the smaller k is. We chose the reset value of k to be 4 instead of 2 because preliminary experiments showed that for $k \in \{2, 3\}$, improvements happened rarely.

The algorithms Win+Swap and MW+Swap additionally search for better solutions with swap distance 1. We limit the algorithms to only search for swap distance 1, since the hardness result of Theorem 3 implies that considering larger swap distances tends to become inefficient. This was also validated by preliminary experiments when considering also better solutions of swap distance 2 and 3. Further, preliminary experiments showed that considering the swap distance provided much better solutions than considering the insert distance. As a consequence, we do not include combinations of insert distance with any of window distance or multi window distance in our evaluation.

Lines 4–7 of Algorithm 1 check for a better schedule in

the 1-swap-neighborhood. Omitting these lines yields our second algorithm: Win. By replacing in Line 8 and Line 11 the window distance by the multi window distance yields MW+Swap. The fourth local search algorithm (MW) is obtained by performing both these operations, that is, omitting Lines 4–7 and replacing in Line 8 and Line 11 the window distance by the multi window distance.

Tardiness and starting solutions. We next describe the starting solutions we used for our algorithms. Unfortunately, providing a feasible starting solution can not be done in polynomial time, unless $\mathrm{P} = \mathrm{NP}$. Hence, in our experimental evaluation, we also allow non-feasible schedules. Then, our goal is to minimize the total tardiness and the makespan, where minimizing the total tardiness is the primary objective. That is, a schedule π' is *better* than a schedule π, if a) the total tardiness of π' is smaller than the total tardiness of π or b) π and π' have the same total tardiness and the makespan of π' is smaller than the makespan of π. Note that under this modified objective function, if the current solution π has total tardiness zero, the algorithms for all distance measures are still correct. If the total tardiness of π is non-zero, the previous described algorithms for finding better solution with window distance or multi window distance are not necessarily correct anymore, that is, the described algorithms output that there is no better solution with distance at most k even though there might be a better solution with distance at most k, where the corresponding change is not EDDS. In our evaluation, we show that even though some local improvements might not be found, the approaches still perform very well when also aiming to minimize the total tardiness also for non-feasible solutions.

We consider three starting solutions for our algorithms. All these starting solutions are computed greedily and are based on the idea of EDDS. The first starting solution (DD) sorts all jobs according to their deadline. The other two solutions (SM and TM) group the jobs by their type, sort the jobs of each type according to their deadline, and find a good order of the types to connect the partial schedules of the individual types. In SM we take an ordering of the types that minimize the total setup times among all such orderings. In TM we take an ordering of the types that minimizes the total tardiness among all such orderings. The latter two starting solution can be computed in $t! \cdot n + n \cdot \log(n)$ time, where t denotes the number of types and n denotes the number of jobs. Since we assume that t is a very small constant (in our experiments, $t = 8$) these starting solutions can be computed efficiently.

Genetic algorithm

As a baseline, we additionally consider two genetic algorithms which maintain a population of size $p = 100$ and consist of the operators Decode (D_e), Crossover (C_o), Mutation (M_u), Evaluate (E_v), and Selection (S_e). We propose a basic genetic algorithm (GAD) and a modified genetic algorithm (MGA). We derived the idea for the MGA based on the Observation 1. It differs from the GAD in the operators D_e, C_o and M_u. To evaluate the quality of the results, we define our *fitness function* as $\Phi(\pi) := C_{\max} + \mu T$, where T is the total tardiness of π, with $\mu := n \cdot (\max_j t_j + \max_{i,j} \ell(\tau_i, \tau_j)) + 1$. Note that $\Phi(\pi)$ essentially models a lexicographic optimization, where the primary goal is to minimize the total tardiness and the secondary goal is to minimize the setup time.

In case of the GAD, we start with an initial solution π and identify each job by a unique number. We then generate p permutations π', which are all EDDS. The operator D_e is used to encode all permutations by storing sequences of the unique numbers. Based on preliminary experiments, we chose a two point crossover method for C_o and a single swap method for M_u. To evaluate, all sequences are decoded and scored with $\Phi(\pi)$. In the last step, the $p - 1$ best schedules with respect to $\Phi(\pi)$ are selected in the operator S_e and the best schedule of the last iteration is copied.

In case of the MGA, we generate the starting population in an identical way but store the type τ_j of each job instead of the unique number. We decode by mapping the τ_j and the relative position to the individual jobs grouped by τ_j and sorted in non-decreasing manner by deadline d_j. With this decoding, all solutions are EDDS. In addition, we adapt C_o by using a type based crossover. In contrast to randomly choosing a position, we randomly select a set, smaller than k, from the given types. Then, all types that occur in the set are copied from the second parent into a new sequence and the empty spaces are filled with all other types from the first parent maintaining their relative order. For the operator M_u, we randomly choose two indices and shuffle the subsequence lying in between. In the final step, the sequences are decoded as described above and then scored in the same way as for the GAD.

Evaluation

We implemented the above mentioned algorithms in Python and evaluated their performance. The experiments were run with Python 3.10.12 on a single thread of an Intel(R) Core(TM) i7-1255U CPU with 4.7 GHz and 16 GB RAM.

The considered data and setup mapping. We perform experiments to evaluate our approaches for the real-world use case from the company X. The products produced by X come in eight different color/material combinations, all of which are manufactured on the same injection molding machine. For our experiments, we use the 8×8 setup matrix containing the machines setup times in minutes as integers.

Since Xs web shop has not yet launched, the total number of real-world orders is relatively small at this time. To evaluate our algorithms on large instances, we adapted well-established benchmark datasets for single-machine scheduling with deadlines established by Tanaka et al. 2009[2]. More precisely, for each job consisting of a processing time and a deadline, we chose a product type from the interval $[1, 8]$ independently at random from an equal distribution. Furthermore, we multiplied the processing time and the deadline of each job by a constant of 50, since otherwise the real-world setup times from X always dominate the solution's makespan by far, which is not realistic in our use case.

[2]https://sites.google.com/site/shunjitanaka/sips/benchmark-results-sips

Table 1: Experimental results. Table 1) displays the setup times of the best feasible schedules that were found within the time limit. In cells marked with †, no feasible schedule was found within the time limit. Table 2) provides an overview on the tardiness of the best schedule found within the time limit. For sake of readability, the table displays $\lfloor T \cdot 10^{-3} \rfloor$ for every computed tardiness T. The best found solution(s) of each instance are marked with grey cell color. Italic numbers indicate that the stated total tardiness was obtained only by 1-swaps and no other local neighborhood was considered within the time limit.

1) dataset: feasible schedule possible

		50	100	150	200	250	300
Win	DD	735	1105	1910	4185	3325	3655
	SM	695	1095	2740	†	1450	750
	TM	695	1360	2170	†	1675	2150
Win+Sw	DD	740	1075	2750	4270	2905	3465
	SM	705	1285	2920	4105	1990	1975
	TM	710	1325	2660	5515	1995	1725
MW	DD	1035	2280	4090	6085	6775	9195
	SM	785	1645	†	†	2245	†
	TM	905	2030	†	†	3305	3330
MW+Sw	DD	1000	1265	3785	5420	4065	6950
	SM	970	1635	3005	4960	2895	2075
	TM	710	1570	3440	5990	2240	2115
GAD		3695	12970	18810	†	32850	46130
MGA		1125	2810	5870	9540	10740	15250

2) dataset: feasible schedule impossible

		50	100	150	200	250	300
Win	DD	889	8429	15213	575	15816	54312
	SM	889	8420	15357	871	15440	53662
	TM	890	8428	15093	761	16129	53824
Win+Sw	DD	730	7296	10774	500	*12526*	*42533*
	SM	731	7251	10845	474	*15331*	*45551*
	TM	735	7276	10798	490	*14812*	*43941*
MW	DD	953	8509	15531	640	16577	56473
	SM	946	8777	15977	1162	19150	58981
	TM	910	8582	15733	897	18517	57543
MW+Sw	DD	736	7323	10848	523	*12526*	*42533*
	SM	731	7308	10919	478	*15331*	*45551*
	TM	732	7316	10865	489	*14812*	*43941*
GAD		810	8039	12823	1247	18512	48864
MGA		904	8745	15975	699	17185	58540

We consider two sets of data. One dataset, where a feasible schedule exists, and one dataset, where no feasible schedule exists. Recall that, in the second case, we then aim to minimize the total tardiness. In both datasets, we have instances with $n \in \{50, 100, 150, 200, 250, 300\}$ jobs.

The concrete tested datasets are the following:

n	dataset $T = 0$	dataset $T > 0$
50	wt050_078	wt050_012
100	wt100_105	wt100_073
150	wt150_108	wt150_017
200	wt200_106	wt200_005
250	wt250_088	wt250_076
300	wt300_125	wt300_052

Experimental Results. For all three starting solutions (DD, SM, TM), we evaluated our four hill climbing strategies (Win, Win+SW, MW, MW+SW) on 1) a dataset of instances where feasible schedules exist, and 2) a dataset of instances where no feasible schedule exists. All with a time limit of 15 minutes. Furthermore, as a baseline, we ran experiments with the two genetic algorithms GAD and MGA. For both genetic algorithms we performed 20 independent repetitions with a time limit of 15 minutes each and kept the best found results. Table 1 provides an overview.

We summarize our main experimental findings as follows. First, all hill climbing algorithm provide good results for instances where feasible schedules are possible. The algorithms were mostly able to provide good results even if the starting solution was not feasible. Overall, the window algorithm appears to be well suited to handle these instances. Second, on instances where no feasible schedules exist, local search via single swaps often has the biggest influence on the solution quality and provides good results. Especially for the large instances with $n \in \{250, 300\}$, all improvements were reached by swaps and no parameterized local search step has been applied within the time limit of 15 minutes. On the smaller instances, the VNS algorithms Win+Sw and MW+Sw help the local search via swaps to escape bad swap-optimal solutions and provide good results for this case. Third, the hill climbing algorithms outperform the baseline genetic algorithms in the solution quality. However, for non-feasible start solutions in the first dataset, MGA was mostly able to find a feasible solution significantly faster (in less then 10 seconds) than the hill climbing algorithms.

Conclusion. Motivated by a real-world use case, we initiated the study of parameterized local search heuristics for single-machine scheduling from a theoretical and practical point of view. Our experimental evaluations indicate that parameterized local search is a promising technique in the design of heuristics for the $1 \mid \mathrm{ST}_{\mathrm{sd,f}} \mid C_{\max}$ scheduling problem and that its usefulness should be explored for further (single machine) scheduling problems.

There are many ways to extend our results in future work. Memetic algorithms (Moscato et al. 1989) combine local search strategies with genetic algorithms. How do memetic algorithms work when the local search steps are performed with $k > 1$? Considering the parameterized complexity, it might be interesting to see whether the results in this work can be extended to other target functions than the makespan: is it possible to obtain similar results if one aims to minimize the (weighted) total tardiness or the maximum tardiness?

References

Allahverdi, A. 2015. The third comprehensive survey on scheduling problems with setup times/costs. *European journal of operational research*, 246(2): 345–378.

Bonnet, É.; Iwata, Y.; Jansen, B. M. P.; and Kowalik, L. 2019. Fine-Grained Complexity of k-OPT in Bounded-Degree Graphs for Solving TSP. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA '19)*, vol-

ume 144 of *LIPIcs*, 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Chen, Y.-Y.; Cheng, C.-Y.; Wang, L.-C.; and Chen, T.-L. 2013. A hybrid approach based on the variable neighborhood search and particle swarm optimization for parallel machine scheduling problems—a case study for solar cell industry. *International Journal of Production Economics*, 141(1): 66–78.

Cheng, T.; and Kovalyov, M. 2001. Single machine batch scheduling with sequential job processing. *IIE Transactions*, 33(5): 413–420.

Chiong, R.; and Dhakal, S., eds. 2009. *Natural Intelligence for Scheduling, Planning and Packing Problems*, volume 250 of *Studies in Computational Intelligence*. Springer. ISBN 978-3-642-04038-2.

Cygan, M.; Fomin, F. V.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer. ISBN 978-3-319-21274-6.

Dörnfelder, M.; Guo, J.; Komusiewicz, C.; and Weller, M. 2014. On the parameterized complexity of consensus clustering. *Theoretical Computer Science*, 542: 71–82.

Downey, R. G.; and Fellows, M. R. 2013. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer.

Eddaly, M.; Jarboui, B.; Bouabda, R.; and Rebai, A. 2009. Hybrid estimation of distribution algorithm for permutation flowshop scheduling problem with sequence dependent family setup times. In *Proceedings of the 39th International Conference on Computers & Industrial Engineering (CIE '09)*, 217–220. IEEE.

Gaspers, S.; Gudmundsson, J.; Jones, M.; Mestre, J.; and Rümmele, S. 2019. Turbocharging Treewidth Heuristics. *Algorithmica*, 81(2): 439–475.

Gaspers, S.; Kim, E. J.; Ordyniak, S.; Saurabh, S.; and Szeider, S. 2012. Don't Be Strict in Local Search! In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI '12)*. AAAI Press.

Graham, R. L.; Lawler, E. L.; Lenstra, J. K.; and Kan, A. R. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, 287–326. Elsevier.

Grüttemeier, N.; Komusiewicz, C.; and Morawietz, N. 2021. Efficient Bayesian Network Structure Learning via Parameterized Local Search on Topological Orderings. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI '21)*, 12328–12335. AAAI Press. Full version available at https://doi.org/10.48550/arXiv.2204.02902.

Grüttemeier, N.; Balzereit, K.; Soni, N.; and Bunte, A. 2023. Efficient Production Scheduling by Exploiting Repetitive Product Configurations. In *Proceedings of the 21st IEEE International Conference on Industrial Informatics (INDIN '23), to appear*.

Guo, J.; Hartung, S.; Niedermeier, R.; and Suchý, O. 2013. The Parameterized Complexity of Local Search for TSP, More Refined. *Algorithmica*, 67(1): 89–110.

Hartung, S.; and Niedermeier, R. 2013. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science*, 494: 86–98.

Held, M.; and Karp, R. M. 1962. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, 10(1): 196–210.

Katzmann, M.; and Komusiewicz, C. 2017. Systematic Exploration of Larger Local Search Neighborhoods for the Minimum Vertex Cover Problem. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI '17)*, 846–852. AAAI Press.

Komusiewicz, C.; Linz, S.; Morawietz, N.; and Schestag, J. 2023. On the Complexity of Parameterized Local Search for the Maximum Parsimony Problem. In *34th Annual Symposium on Combinatorial Pattern Matching, CPM 2023*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. Accepted for publication.

Liao, C.; and Juan, H. 2007. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. *Computers & Operations Research.*, 34(7): 1899–1909.

Marx, D. 2008. Searching the k-change neighborhood for TSP is W[1]-hard. *Operations Research Letters*, 36(1): 31–36.

Moscato, P.; et al. 1989. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826(1989): 37.

OuYang, Q.; and Xu, H. Y. 2014. Genetic Algorithm for single machine scheduling problem with setup times. *Applied Mechanics and Materials*, 457: 1678–1681.

Rego, M. F.; Souza, M. J. F.; and Arroyo, J. E. C. 2012. Multi-objective Algorithms for the Single Machine Scheduling Problem with Sequence-Dependent Family Setups. In *Proceedings of the 31st International Conference of the Chilean Computer Science Society, (SCCC '12)*, 142–151. IEEE Computer Society.

Riahi, V.; Newton, M. A. H.; and Sattar, A. 2021. Constraint based local search for flowshops with sequence-dependent setup times. *Engineering Applications of Artificial Intelligence*, 102: 104264.

Riahi, V.; Newton, M. A. H.; Su, K.; and Sattar, A. 2018. Local Search for Flowshops with Setup Times and Blocking Constraints. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling, (ICAPS '18)*, 199–207. AAAI Press.

Szeider, S. 2011. The parameterized complexity of k-flip local search for SAT and MAX SAT. *Discrete Optimization*, 8(1): 139–145.

Tanaka, S.; Fujikuma, S.; and Araki, M. 2009. An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling*, 12(6): 575–593.