The Parameterized Complexity of Cascading Portfolio Scheduling (Full Version)

Anonymous Author(s)
Affiliation
Address
email

Abstract

Cascading portfolio scheduling is a static algorithm selection strategy which uses a sample of test instances to compute an optimal ordering (a cascading schedule) of a portfolio of available algorithms. The algorithms are then applied to each future instance according to this cascading schedule, until some algorithm in the schedule succeeds. Cascading scheduling has proven to be effective in several applications, including QBF solving and generation of ImageNet classification models.

It is known that the computation of an optimal cascading schedule in the offline phase is NP-hard. In this paper we study the parameterized complexity of this problem and establish its fixed-parameter tractability by utilizing structural properties of the success relation between algorithms and test instances. Our findings are significant as they reveal that in spite of the intractability of the problem in its general form, one can indeed exploit sparseness or density of the success relation to obtain non-trivial runtime guarantees for finding an optimal cascading schedule.

1 Introduction

When dealing with hard computational problems, one often has access to a *portfolio* of different algorithms that can be applied to solve the given problem, with each of the algorithms having complementary strengths. There are various ways of how this performance complementarity can be exploited. Algorithm selection, a line of research initiated by Rice [21], studies various approaches one can use to select algorithms from the portfolio. Algorithm selection has proven to be an extremely powerful tool with many success stories in Propositional Satisfiability, Constraint Satisfaction, Planning, QBF Solving, Machine Learning and other domains [13, 14, 15, 22]. A common approach to algorithm selection is *per-instance-based algorithm selection*, where an algorithm is chosen for each instance independently, based on some features of the instance (see, *e.g.*, [16, 11]). However, sometimes information about the individual instances is not available or difficult to use. Then, one can instead make use of information about the distribution of the set of instances, *e.g.*, in terms of a representative sample of instances which can be used as a *training set*. In such cases, one can compute in an offline phase a suitable linear ordering of the algorithms, optimizing the ordering for the training set of instances. This ordering is then applied uniformly to any given problem instance in an online fashion—in particular, if the first algorithm in our ordering fails to solve a given instance (due to timeout, memory overflow, or due to not reaching a desired accuracy), then the second algorithm is called, and this continues until we solve the instance. Such a static algorithm selection, "*cascading portfolio scheduling*", is simpler to implement than per-instance selection methods and can be very effective [24]. One prominent recent application of cascading portfolio scheduling lies in state-of-the-art *ImageNet classification models*, where it resulted in a significant speedup by reducing the number of floating-point operations [25]. Cascading portfolio scheduling is also related to online portfolio scheduling [12, 17].

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.

In this paper we address the fundamental problem of finding an optimal cascading schedule for a given portfolio $\mathcal{A}$ of algorithms with respect to a given training set T of instances. In particular, for the problem CASCADING PORTFOLIO SCHEDULING (or CPS for short) that we consider, we are given m algorithms, n test instances, and a *cost mapping* cost, where $\text{cost}(\alpha, t)$ denotes the cost of running algorithm α on test instance t, and a *success relation* S where $(\alpha, t) \in S$ means that algorithm α succeeds on test instance t. As the cost mapping and the success relation are defined independently, this setting is very general and entails different scenarios.

Scenario 1 Each algorithm is run until a globally set timeout C is reached. If the algorithm α solves test instance t in time $c \leq C$ then $\text{cost}(\alpha, t) = c$ and $(\alpha, t) \in S$; otherwise we have $\text{cost}(\alpha, t) = C$ and $(\alpha, t) \notin S$.

Scenario 2 Algorithm α solves a test instance t in time c and outputs an accuracy estimate r for its solution. r is then compared with a globally set accuracy threshold R. If $r \geq R$ then $(\alpha, t) \in S$, otherwise $(\alpha, t) \notin S$; in any case $\text{cost}(\alpha, t) = c$. Such a strategy has been used for prediction model generation [25].

Scenario 3 All the algorithms are first run with a short timeout and if the test instance has not been solved after this, algorithms are run again without a timeout (a similar strategy has been used for QBF solving [19]). Such a strategy can be instantiated to our setting by adding two copies of each algorithm to the portfolio, one with a short timeout and one without a timeout.

Contribution. We establish the *fixed-parameter tractability*[1] of computing an optimal cascading schedule by utilizing structural properties of the success relation. We look at the success relation in terms of a Boolean matrix, the *evaluation matrix*, where each row corresponds to a test instance and each column corresponds to an algorithm. A cell contains the entry 1 iff the corresponding algorithm succeeds on the corresponding test. We show that if this matrix is either very sparse or very dense, then the computation of an optimal schedule is tractable. More specifically, we establish the following results, which we describe by writing CPS[*parm*] for CASCADING PORTFOLIO SCHEDULING parameterized by parameter *parm*.

First we consider the *algorithm failure degree* which is the largest number of tests a single algorithm fails on, and the *test failure degree* which is the largest number of algorithms that fail on a single test (these two parameters can also be seen as the largest number of 0's that appear in a row and the largest number of 0's that appear in a column of the matrix, respectively).

(1) CPS[*algorithm failure degree*] and CPS[*test failure degree*] are fixed-parameter tractable (Theorems 4 and 5).

It is natural to consider also the dual parameters *algorithm success degree* and *test success degree*. However, it follows from known results that CPS is already NP-hard if both of these parameters are bounded by a constant (Proposition 6). Hence, our results exhibit a certain asymmetry between failure and success degrees.

We then consider more sophisticated parameters that capture the sparsity or density of the evaluation matrix. The *failure cover number* is the smallest number of rows and columns in the evaluation matrix needed to cover all the 0's in the matrix; similarly, the *success cover number* is the smallest number of rows and columns needed to cover all the 1's. In fact, both parameters can be computed in polynomial time using bipartite vertex cover algorithms [8].

(2) CPS[*failure cover number*] and CPS[*success cover number*] are fixed-parameter tractable (Corollary 8 and Theorem 19).

These results are significant as they indicate that CASCADING PORTFOLIO SCHEDULING can be solved efficiently as long as the evaluation matrix is sufficiently sparse or dense. Our result for CPS[*failure cover number*] in fact also shows fixed-parameter tractability of the problem for an even more general parameter than success cover number: the treewidth [23] of the bipartite graph between the algorithms and tests, where edges join success pairs. This is our most technical contribution and reveals how a fundamental graphical parameter [see, *e.g.*, 9] can be utilized for algorithm scheduling.

[1] Fixed-parameter tractability is a relaxation of polynomial tractability; definitions are provided in Section 2.

Another natural variant of the problem, CPS$^{\text{opt}}$[*length*], arises by adding an upper bound ℓ on the length, *i.e.*, cardinality, of the computed schedule, and asking for a schedule of length $\leq \ell$ of minimum cost. We obtain a complexity classification of the problem under this parameterization as well.

(3) CPS[*length*] can be solved in polynomial time for each fixed bound ℓ, but is not fixed-parameter tractable parameterized by ℓ subject to established complexity assumptions.

2 Preliminaries

Problem Definition. An instance of the CASCADING PORTFOLIO SCHEDULING problem is a tuple $(\mathcal{A}, T, \text{cost}, S)$ comprising:

- a set $\mathcal{A}$ of m algorithms,
- a set T of n tests,
- a cost mapping cost : $(\mathcal{A} \times T) \to \mathbb{N}$, and
- a success relation $S \subseteq \mathcal{A} \times T$.

Let τ be a totally ordered subset of $\mathcal{A}$; we call such a set a *schedule*. The *length* of a schedule is its cardinality. We say that τ is *valid* if for each test t there exists an algorithm $\alpha \in \tau$ such that $(\alpha, t) \in S$. Throughout the paper, we will assume that there exists a valid schedule for our considered instances—or, equivalently, that each test is solved by at least one algorithm.

The *processing cost* of a test t for a valid schedule $\tau = (\alpha_1, \ldots, \alpha_q)$ is defined as $\sum_{i=1}^{j} \text{cost}(\alpha_i, t)$, where j is the *first* algorithm in τ such that $(\alpha_j, t) \in S$. The *cost* of a valid schedule τ, denoted cost(τ), is the sum of the processing costs of all tests in T for τ. The aim in CASCADING PORTFOLIO SCHEDULING is to find a valid schedule τ of minimum cost.

Parameterized Complexity. In parameterized algorithmics [7, 5, 4, 10] the complexity of a problem is studied not only with respect to the input size n but also a parameter $k \in \mathbb{N}$. The most favorable complexity class in this setting is FPT (*fixed-parameter tractable*) which contains all problems that can be solved by an algorithm running in time $f(k) \cdot n^{\mathcal{O}(1)}$, where f is a computable function. Algorithms running in this time are called *fixed-parameter algorithms*. We will also make use of the complexity classes W[2] and XP, where W[2] $\subseteq$ XP. Problems complete for W[2] are are widely believed to not be in FPT. The class XP contains problems that are solvable in time $\mathcal{O}(n^{f(k)})$, where f is a computable function; in other words, problems in XP are polynomial-time solvable when the parameter is bounded by a constant. To obtain our lower bound results, we will need the notion of a parameterized reduction, referred to as FPT-*reduction*, which is in many ways analogous to the standard polynomial-time reductions; the distinction is that a parameterized reduction runs in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f, and provides upper bounds on the parameter size in the resulting instance [5, 4, 7, 18].

We write $\mathcal{O}^*(f(k))$ to denote a function of the form $f(k) \cdot n^{\mathcal{O}(1)}$, where n is the input length and k is the parameter.

Problem Parameters. CASCADING PORTFOLIO SCHEDULING is known to be NP-hard [25], and our aim in this paper will be to circumvent this by obtaining parameters that exploit the fine-grained structure in relevant problem instances. We note that we explicitly aim for results which allow for arbitrary cost mappings, since these are expected to consist of large (and often disorderly) numbers in real-life settings. Instead, we will consider parameters that restrict structural properties of the "binary" success relation. To visualize this success relation, it will be useful to view an instance $\mathcal{I}$ as an $m \times n$ matrix $M_{\mathcal{I}}$ where $M_{\mathcal{I}}[i, j] = 1$ if $(\alpha_i, t_j) \in S$ (*i.e.* if the j-th test succeeds on the i-th algorithm, for some fixed ordering of algorithms and tests), and $M_{\mathcal{I}}[i, j] = 0$ otherwise.

The two most natural parameters to consider are m and n, and these correspond to the number of rows and columns in $M_{\mathcal{I}}$, respectively. Unfortunately, these two parameters are also fairly restrictive—it is unlikely that instances of interest will have a very small number of algorithms or test instances. Another option would be to use the maximum number of times an algorithm (or test) can fail (or succeed) as a parameter. In particular, the *algorithm success (or failure) degree* is the maximum number of 1's (or 0's, respectively) occurring in any row in $M_{\mathcal{I}}$. Similarly, we let the *test success (or failure) degree* be the maximum number of 1's (or 0's, respectively) occurring in any column in $M_{\mathcal{I}}$. Instances where these parameters are small correspond to cases where "almost everything" either fails or succeeds.

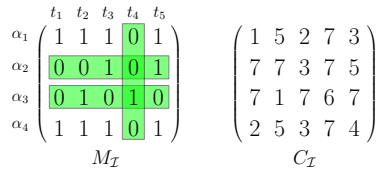

Figure 1: An instance with 4 algorithms and 5 tests in the setting where (exact) algorithms are executed with a global timeout of 7, as discussed in Scenario 1. On the left is the matrix $M_{\mathcal{I}}$ representing the success relation. The failure covering number is 3, as witnessed by the highlighted two rows and one column. The matrix $C_{\mathcal{I}}$ on the right represents the cost relation, with $C_{\mathcal{I}}[i,j] = \text{cost}[\alpha_i, t_j]$. The instance $\mathcal{I}$ depicted here has a single solution, notably (α_1, α_3).

A more advanced parameter that can be extracted from $M_{\mathcal{I}}$ is the covering number, which intuitively captures the minimum number of rows and columns that are needed to "cover" all successes (or failures) in the matrix. More formally, we say that an entry $M_{\mathcal{I}}[i,j]$ is covered by row i and by column j. Then the *success (or failure) covering number* is the minimum value of $r + c$ such that there exist r rows and c columns in $M_{\mathcal{I}}$ with the property that each occurrence of 1 (or 0, respectively) in $M_{\mathcal{I}}$ is covered by one of these rows or columns. Intuitively, an instance has success covering number s if there exist r algorithms and $s - r$ tests such that these have a non-empty intersection with every relation in S—see Figure 2 for an example. We note that the covering number has been used as a structural parameter of matrices, notably in previous work on the MATRIX COMPLETION problem [8], and that it is possible to compute r algorithms and c tests achieving a minimum covering number in polynomial time [8, Proposition 1]. We will denote the success covering number by cov_s and the failure covering number by cov_f.

3 Results for Basic Parameters

In this section we consider the CASCADING PORTFOLIO SCHEDULING problem parameterized by the *number of algorithms* (*i.e.*, by $m = |\mathcal{A}|$), by the number of tests (*i.e.*, by $n = |T|$), and by the *length* of the computed schedule.

We begin mapping the complexity of our problem with two initial propositions. Note that both propositions can also be obtained as corollaries of the more general Theorem 19, presented later. Still, we consider it useful to present the proof of Proposition 1, since it nicely introduces the combinatorial techniques that will later be extended in the proof of Theorem 1.

Proposition 1. CPS[*number of algorithms*] is in FPT.

Proof. We reduce the problem to that of finding a minimum-weight path in a directed acyclic graph (DAG) D. We construct D as follows. We create a single source vertex s, and a single destination vertex t in D. We define $L_0 = \{s\}$, $L_{m+1} = \{t\}$, and apart from t, D contains m layers, $L_0, \ldots, L_m$, of vertices, where layer L_i, for $i \in \{0, \ldots, m\}$, contains a vertex for each subset of $\mathcal{A}$ of cardinality i, with vertex s corresponding to the empty set. We connect each vertex that corresponds to a subset of $\mathcal{A}$ which is a valid portfolio to t. For each vertex u in layer L_i, $i \in \{0, \ldots, m-1\}$, corresponding to a subset $S_u \subset \mathcal{A}$, and each vertex $v \in L_{i+1}$ corresponding to a subset $S_v \subseteq \mathcal{A}$, where $S_v = S_u \cup \{\alpha\}$, for $\alpha \in \mathcal{A}$, we add an edge (u, v) if there exists a test $t \in T$ such that (1) $(\alpha, t) \in S$ and (2) there does not exist $\beta \in S_u$ such that $(\beta, t) \in S$; in such case the weight of (u, v), $wt(u, v)$, is defined as follows. Let $T_\alpha \subseteq T$ be the set of tests that cannot be solved by any algorithm in S_u. Then $wt(u, v) = \sum_{t \in T_\alpha} cost(\alpha, t)$. Informally speaking, the weight of (u, v) is the additional cost incurred by appending algorithm α to any (partial) portfolio consisting of the algorithms in S_u. This completes the construction of D.

It is not difficult to see that an optimal portfolio for $\mathcal{A}$ corresponds to a minimum-weight path from s to t. More specifically, if $P = (v_0 = s, v_1, \ldots, v_r, t)$ is a minimum-weight s-t path in D, then the schedule $(\alpha_1, \ldots, \alpha_r)$, where $\{\alpha_i\} = S_{v_i} \setminus S_{v_{i-1}}$ (S_{v_i} is the subset of algorithms corresponding to v_i and $S_{v_{i-1}}$ that corresponding to v_{i-1}), for $i \in [r]$, is an optimal schedule for T. The number of vertices in D is $2^m + 1$, the number of edges $\mathcal{O}(2^m m^2)$, and computing a minimum-weight path in D can be done in linear time in the size of D [3], which is $\mathcal{O}^*(2^m)$. It follows that the problem can be solved in time $\mathcal{O}^*(2^m)$. □

Proposition 2. CPS[*number of tests*] is in FPT.

To formally capture the parameterization of the problem by the length ℓ of the computed schedule, we need to slightly adjust its formal definition. Let CPS^{val}[*length*] and CPS^{opt}[*length*] denote the variants of CASCADING PORTFOLIO SCHEDULING where for each problem instance we are also given an integer $\ell > 0$ and only schedules up to length ℓ are considered (ℓ being the parameter). CPS^{val}[*length*] is the decision problem that asks whether there exists a valid schedule of length $\leq \ell$,

and CPS$^{\text{opt}}$[*length*] asks to compute a valid schedule of length $\leq \ell$ of smallest cost or decide that no valid schedule of length $\leq \ell$ exists. Both problems are parameterized by the length ℓ.

Proposition 3. CPS$^{\text{opt}}$[*length*] is in XP, but is unlikely to be in FPT since already CPS$^{\text{val}}$[*length*] is W[2]-complete.

Proof. Membership of CPS$^{\text{opt}}$[*length*] in XP is easy: We enumerate every ordered selection of at most ℓ algorithms from $\mathcal{A}$ (there are at most $\mathcal{O}(\ell! m^{\ell})$ many) and if valid, we compute its cost, and keep track of a valid selection (if any) of minimum cost over all enumerations.

To prove the W[2]-completeness of CPS$^{\text{val}}$[*length*], we give FPT-reductions from and to the W[2]-complete problem SET COVER [5]. The reduction to SET COVER (showing membership in W[2]) is straightforward: The set of tests T is the ground set U of the constructed instance of SET COVER, and for each algorithm $\alpha \in \mathcal{A}$, we add to the family of subsets of U, $\mathcal{F}$, a set $F_\alpha = \{t \in T \mid (\alpha, t) \in S\}$. It is clear that the resulting instance of SET COVER has a cover of size at most ℓ iff there a valid cascading portfolio scheduling of length at most ℓ, and hence the above reduction is an FPT-reduction to SET COVER.

Next, we give an FPT-reduction from SET COVER to our problem. Given an instance $((U, \mathcal{F}), k)$ of SET COVER, where U is a ground set of elements, $\mathcal{F}$ is a family of subsets for U, and $k \in \mathbb{N}$ is the parameter, we create an instance of CASCADING PORTFOLIO SCHEDULING as follows. We set $T = U$, and for each $F \in \mathcal{F}$, we create an algorithm $\alpha_F \in \mathcal{A}$ and add (α_F, t) to S, for every $t \in F$. Finally, we set $\ell = k$. The function *cost* can be defined arbitrarily. The above reduction is clearly a (polynomial-time) FPT-reduction, and it is straightforward to verify that $((U, \mathcal{F}), k)$ is a yes-instance of SET COVER if and only if the constructed instance of CASCADING PORTFOLIO SCHEDULING has a valid portfolio of size at most ℓ. □

4 Results for Degree Parameters

This section presents a classification of the complexity of CASCADING PORTFOLIO SCHEDULING parameterized by the considered (success and failure) degree parameters.

Proposition 4. CPS[*algorithm failure degree*] is in FPT.

Proof. Denote by $\deg_f^{\mathcal{A}}$ the algorithm failure degree, and let $\mathcal{I} = (\mathcal{A}, T, \text{cost}, S)$ be an instance of CASCADING PORTFOLIO SCHEDULING. Consider an algorithm which loops over each algorithm $\alpha \in \mathcal{A}$ and proceeds under the assumption that α is the first algorithm in an optimal valid portfolio. For each such α, the number of tests in T that cannot be evaluated by α is at most $\deg_f^{\mathcal{A}}$. Removing α from $\mathcal{A}$ and the subset of tests $\{t \mid (\alpha, t) \in S\}$ from T results in an instance $\mathcal{I}^-$ of CASCADING PORTFOLIO SCHEDULING with at most $\deg_f^{\mathcal{A}}$ tests, which, by Proposition 2, can be solved in time $\mathcal{O}^*((\deg_f^{\mathcal{A}})^{\deg_f^{\mathcal{A}}})$ to obtain an optimal solution for $\mathcal{I}^-$. Prefixing α to the optimal solution obtained for $\mathcal{I}^-$ (assuming a solution exists) results in an optimal solution S_α for $\mathcal{I}$ under the constraint that algorithm α is the first algorithm. Enumeration every algorithm $\alpha \in \mathcal{A}$ as the first algorithm, computing S_α, and keeping track of the solution of minimum cost over all enumerations, results in an optimal solution for $\mathcal{I}$. The running time of the above algorithm is $\mathcal{O}^*((\deg_f^{\mathcal{A}})^{\deg_f^{\mathcal{A}}})$. □

Proposition 5. CPS[*test failure degree*] is in FPT.

Proof Sketch. Denote by $\deg_f^T$ the test failure degree, and let $\mathcal{I} = (\mathcal{A}, T, \text{cost}, S)$ be an instance of CASCADING PORTFOLIO SCHEDULING. Consider an algorithm which (1) loops over each algorithm $\alpha \in \mathcal{A}$ and proceeds under the assumption that α is the last algorithm in an optimal valid portfolio τ, and then (2) loops over every test t in our instance and proceed under the assumption that t is a test that is solved *only* by α in τ. For each such choice of t and α, it follows that the algorithms preceding α in τ *do not* solve t, and hence there are at most $\deg_f^T$ many such algorithms. Therefore, we can check the validity and compute the cost of every possible ordered selection of a subset from these algorithms that precede α in τ. After we finish looping over all choices of α and t, we output a valid portfolio of minimum cost.

There are $|\mathcal{A}|$ choices for a last algorithm α and $|T|$ choices for a desired test t. For each fixed α and t, there are at most $\mathcal{O}^*((\deg_f^T)!)$ many ordered selections of a subset of algorithms preceding α in τ. It follows that the problem can be solved in time $\mathcal{O}^*((\deg_f^T)!)$. □

Proposition 6. CPS[*algorithm success degree*], CPS[*test success degree*], and even CPS[*algorithm success degree + test success degree*] are NP-hard already if the algorithm success degree is at most 3 and test success degree is at most 2.

Proof. We reduce from the problem 3-MIN SUM VERTEX COVER, where we are given a graph $H = (V, E)$ with maximum degree 3, and the task is to find a bijection $\sigma : V \to \{1, \ldots .V\}$ that minimizes $\sum_{e \in E} f_\sigma(e)$, where $f_\sigma(e) = \min_{v \in e} \sigma(v)$. Feige *et al.* [6] showed that there exists $\epsilon > 0$ such that it is NP-hard to approximate 3-MIN SUM VERTEX COVER within a ratio better than $1 + \epsilon$. Given an instance of this problem, we construct an instance of $(\mathcal{A}, T, \text{cost}, S)$ of CASCADING PORTFOLIO SCHEDULING by letting $\mathcal{A} = V$, adding for each edge $e \in E$ a test t_e to T, setting $S = \{ (\alpha, t_e) \in \mathcal{A} \times T : \alpha \in e \}$, and setting $\text{cost}(\alpha, t) = 1$ for all $\alpha \in \mathcal{A}$ and $t \in \mathcal{T}$. It is easy to verify that bijections σ that minimize $\sum_{e \in E} f_\sigma(e)$ are exactly those that give an ordering τ of $\mathcal{A}$ of minimal cost. It remains to observe that the the algorithm success degree is 3 and the test success degree is 2. □

5 Results for Cover Numbers

In this section we show that CPS[*failure cover number*] and CPS[*success cover number*] are both fixed-parameter tractable.

5.1 Using the Failure Cover Number

The first of the two results follows from an even more general result, the fixed-parameter tractability of CPS[*failure treewidth*], where as the parameter we take the *treewidth* of the failure graph $G_\mathcal{I}$ defined as follows.

The failure graph $G_\mathcal{I}$ is a bipartite graph whose vertices consist of $\mathcal{A} \cup T$ and where there is an edge between $\alpha \in \mathcal{A}$ and $t \in T$ iff t fails on $\mathcal{A}$. We note that the algorithm (or test) failure degree naturally corresponds to the maximum degree in the respective bipartition of $G_\mathcal{I}$, and that the failure covering number is actually the size of a minimum vertex cover in $G_\mathcal{I}$.

Treewidth [23, 9, 1] is a well-established graph parameter that measures the "tree-likeness" of instances. Aside from treewidth, we will also need the notion of *balanced separators* in graphs. We introduce these technical notions below.

Treewidth and Separators. Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $(\mathcal{V}, \mathcal{T})$ where $\mathcal{V}$ is a collection of subsets of V such that $\bigcup_{X_i \in \mathcal{V}} = V$, and $\mathcal{T}$ is a rooted tree whose node set is $\mathcal{V}$, such that:

1. For every edge $\{u, v\} \in E$, there is an $X_i \in \mathcal{V}$, such that $\{u, v\} \subseteq X_i$; and
2. for all $X_i, X_j, X_k \in \mathcal{V}$, if the node X_j lies on the path between the nodes X_i and X_k in the tree $\mathcal{T}$, then $X_i \cap X_k \subseteq X_j$.

The *width* of the tree decomposition $(\mathcal{V}, \mathcal{T})$ is defined to be $\max\{|X_i| \mid X_i \in \mathcal{V}\} - 1$. The *treewidth* of the graph G, denoted $\text{tw}(G)$, is the minimum width over all tree decompositions of G. A tree decomposition $(\mathcal{V}, \mathcal{T})$ is *nice* if it satisfies the following conditions:

1. Each node in the tree $\mathcal{T}$ has at most two children.
2. If a node X_i has two children X_j and X_k in the tree $\mathcal{T}$, then $X_i = X_j = X_k$; in this case node X_i is called a *join node*.
3. If a node X_i has only one child X_j in the tree $\mathcal{T}$, then either $|X_i| = |X_j| + 1$ and $X_j \subset X_i$, and in this case X_i is called an *insert node*; or $|X_i| = |X_j| - 1$ and $X_i \subset X_j$, and in this case X_i is called a *forget node*.
4. If X_i is a leaf node or the root, then $X_i = \emptyset$.

A pair of vertex subsets (A, B) is a *separation* in graph G if $A \cup B = V(G)$ and there is no edge between $A \setminus B$ and $B \setminus A$. The *separator* of this separation is $A \cap B$, and the *order* of separation (A, B) is equal to $|A \cap B|$. We say that a separation (A, B) of G is an α-*balanced* separation if $|A \setminus B| \leq \alpha |V(G)|$ and $|B \setminus A| \leq \alpha |V(G)|$.

Proof Strategy. Our main aim in this section will be to prove the following theorem:

Theorem 7. CPS[*failure treewidth*] is in FPT.

It is easy to see that failure treewidth is at most the failure cover number plus 1 (consider, *e.g.*, a tree decomposition of the failure graph consisting of a sequence of bags, each containing the algorithms and tests forming the cover and one additional test or algorithm). Hence, once we establish Theorem 7 we obtain the following as an immediate corollary:

Corollary 8. CPS[*failure cover number*] is in FPT.

The proof of Theorem 7 is broken into two main steps, and we provide a high-level overview of these below.

In the first step, we show that there exists an optimal solution of size at most $\mathcal{O}(tw(G_{\mathcal{I}}) \cdot \log(m+n))$. To argue this property, we consider a balanced separator B of $G_{\mathcal{I}}$ of size $tw(G_{\mathcal{I}})+1$, and assume w.l.o.g. that, in an optimal solution, an algorithm on the left side of B occurs before an algorithm on the right side of B. It can be shown that there exists an optimal solution which behaves as follows: the first algorithm located on the right side of B is followed by at most $tw(G_{\mathcal{I}})+1$ other algorithms. With this insight, we can show that $tw(G_{\mathcal{I}})+3$ many algorithms solve at least a constant fraction of all tests in the instance. Applying this argument recursively allows us to obtain the desired bound on the size of the optimal solution.

In the second step, we solve the problem using dynamic programming on a tree decomposition of $G_{\mathcal{I}}$, by utilizing the upper bound on the solution length derived in the first step. The running time is $\mathcal{O}^*(4^{\ell} \cdot \ell^{\mathrm{tw}(G_{\mathcal{I}})} \cdot \mathrm{tw}(G_{\mathcal{I}})!)$. To make the dynamic programming approach work, for a current bag in the tree decomposition, and for each position in the schedule, we remember whether this position will be filled by an algorithm from the "future" (*i.e.*, has not been seen yet), whether it was already filled in the "past", or whether it will be filled by an algorithm from the current bag. We also remember specifically which algorithm is the "first" from the future, which is the "first" from the past, and what are the positions of the algorithms in the bag. Moreover, for each test in the bag, we remember the position of the algorithm that solves the test.

The Proof. We start with the following lemma:

Lemma 9. *There exists a minimum cost schedule for* CASCADING PORTFOLIO SCHEDULING *with length at most* $(2tw(G_{\mathcal{I}})+5) \cdot \log(m+n)$.

Proof. As our starting point, we establish the following claim concerning *proper valid schedules*, which are schedules where each algorithm solves at least one test that has not been solved by previous algorithms.

Claim 10. *Let (A,B) be a separation in $G_{\mathcal{I}}$ with separator $X = A \cap B$. Then in every proper valid schedule, either all the algorithms from $A \setminus X$ or all the algorithms from $B \setminus X$ are among the last $|X \cap T|+1$ scheduled algorithms.*

Proof of Claim. Let τ be such a valid schedule. Assume, w.l.o.g., that an algorithm $\alpha_A \in A \setminus X$ is scheduled before any algorithm from $B \setminus X$, and let $\alpha_B \in B \setminus X$ be the first algorithm from $B \setminus X$ in τ. Since (A,B) is a separation in the failure graph with separator X, every algorithm in $A \setminus X$ solves every test in $B \setminus X$ and vice versa. Hence, it is easy to see that the only tests that may not have been solved when τ passes α_B are tests in X. Since in τ every algorithm solves at least one new test, there are at most $|X \cap T|$ algorithms scheduled after α_B, and the claim follows. □

Let $\mathcal{I} = (\mathcal{A}, T, \mathrm{cost}, S)$ be an instance of CASCADING PORTFOLIO SCHEDULING. We will prove the lemma by induction on $|\mathcal{A}|+|T|$ and by restricting our attention to proper valid schedules only; the latter is justified by the fact that every valid schedule can be turned into a proper valid schedule without increasing its cost. Clearly, if $|\mathcal{A}|+|T| \leq 2$ then the lemma holds.

Now let us assume inductively, for every instance $\mathcal{I}' = (\mathcal{A}', T', \mathrm{cost}', S')$ with $|\mathcal{A}'|+|T'| < |\mathcal{A}|+|T|$, that every proper valid schedule has length at most $(2tw(G_{\mathcal{I}'}+5) \cdot \log(|\mathcal{A}'|+|T'|)$. Let τ be such a valid schedule for $\mathcal{I}$, and let (A,B) be a $\frac{2}{3}$-balanced separation in $G_{\mathcal{I}}$ with separator X of size at most $tw(G_{\mathcal{I}})+1$, it is well known that such separation always exists (see, *e.g.*, [4, Lemma 7.20]). If the length of τ is at most $2|X|+3$, then we are done. Otherwise, the sets A, B, X satisfy the conditions of Claim 10, and hence all the algorithms from one of the parts, say $B \setminus X$, are among the last $|X \cap T|+1$ scheduled algorithms. Moreover, since τ contains more than $2|X|+3$ algorithms, there is an algorithm α_A that is the first algorithm from $A \setminus X$ in τ. Now let $\mathcal{I}' = (\mathcal{A}', T', \mathrm{cost}', S')$ be the instance obtained from $\mathcal{I}$ by removing:

- α_A and all tests solved by α_A,
- all tests and algorithms in $B \setminus X$, and
- all tests solved in τ by some algorithm in $B \setminus X$ and no algorithm before them in the schedule.

Note that $|\mathcal{A}'| + |T'| \leq \frac{2}{3}(|\mathcal{A}| + |T|)$. Now let τ' be the schedule obtained from τ by removing α_A, all algorithms from B, and all algorithms from X that appear before α_A and only succeed on tests in B. Note that we removed at most $1+|X\cap T|+1+|X\cap\mathcal{A}| = |X|+2$ algorithms from τ. Now, since after α_A all the tests in $B \setminus X$ were solved and every algorithm in τ solves at least one new test (*i.e.*, a test not solved by previous algorithms), it follows that every algorithm in τ' solves at least one new test. Moreover, we removed from T' all the tests that were solved in τ by α_A or an algorithm in B and hence τ' is a valid schedule for $\mathcal{I}'$. Finally, note that $tw(G_{\mathcal{I}'}) < tw(G_{\mathcal{I}})$. Hence by our assumption the length of τ' is at most $(2tw(G_{\mathcal{I}})+5)\log(|\mathcal{A}'|+|T'|) \leq (2tw(G_{\mathcal{I}})+5)\log(\frac{2}{3}(|\mathcal{A}|+|T|)) = (2tw(G_{\mathcal{I}})+5)\log(|\mathcal{A}|+|T|) - (2tw(G_{\mathcal{I}})+5)\log(\frac{3}{2})$. The lemma then follows by the fact that $tw(G_{\mathcal{I}})+3 < (2tw(G_{\mathcal{I}})+5)\log(\frac{3}{2})$ for $tw(G_{\mathcal{I}}) \geq 1$. □

This concludes the first step towards the proof of Theorem 7. For the second step, we need to show the following lemma:

Lemma 11. *A minimum cost schedule for* CASCADING PORTFOLIO SCHEDULING *among the schedules of length exactly ℓ can be computed in time $\mathcal{O}^*(4^\ell \cdot \ell^{\mathrm{tw}} \cdot \mathrm{tw}!)$.*

Proof Sketch. As with virtually all fixed-parameter algorithms parameterized by treewidth, we use leaf-to-root dynamic programming along a nicetree decomposition (in this case of the failure graph $G_{\mathcal{I}}$)—see for instance the numerous examples presented in the literature [5, 4]. However, due to the specific nature of our problem, the records dynamically computed by the program are far from standard. This can already be seen by considering the size of our records: while most such dynamic programming algorithms only store records that have size bounded by a function of the treewidth, in our case the records will also have a polynomial dependence on m.

As a starting point, we will use the known algorithm of Bodlaender *et al.* [2] to compute a nicetree-decomposition of width at most $5 \cdot \mathrm{tw}(G_{\mathcal{I}})$. We proceed by formalizing the used records. Let X_i be a bag in the tree decomposition. A *configuration* w.r.t. X_i is a tuple $(\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma, \delta)$, where

- α_{past} is an algorithm that has been forgotten in a descendant of X_i,
- α_{future} is an algorithm that has not been introduced yet in X_i,
- $\sigma : [\ell] \to \{\text{“past”}, \text{“future”}\} \cup (\mathcal{A} \cap X_i)$, and
- $\delta : T \cap X_i \to [\ell]$.

The interpretation of the configuration is that $\sigma[j]$ tells us whether the j-th algorithm in the final schedule has been already introduced and forgotten (“past”), it is a specific algorithm in the bag X_i, or it has not been introduced yet (“future”). The function δ, for a test t, tells us that t is solved by the $\delta[t]$-th algorithm and not solved by any algorithm before. The entries α_{past} and α_{future} represent the specific algorithms with the lowest indices j, j' with $\sigma[j] =$ “past” and $\sigma[j'] =$ “future”, respectively. We will refer to the index j such that $\sigma[j] =$ “past” for all $j' < j$ is $\sigma[j'] \neq$ “past” as *the index of the first “past”* and, similarly, to the index j such that $\sigma[j] =$ “future” and for all $j' < j$ is $\sigma[j'] \neq$ “future” as *the index of the first “future”*. Note that the indices of the first “past” and the first “future”, respectively, are interpreted as precisely the positions of α_{past} and α_{future} in the schedule.

We say that a configuration $C = (\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma, \delta)$ w.r.t. X_i is *admissible* if

- for each algorithm $\alpha \in \mathcal{A} \cap X_i$, there exists precisely one index j such that $\sigma[j] = \alpha$;
- if $\delta[t] = j$, then for every $j' < j$:
  - if $\sigma[j'] \in \mathcal{A} \cap X_i$ then $\sigma[j']$ does not solve t, and
  - if $\sigma[j'] =$ “past” (resp. $\sigma[j'] =$ “future”), then α_{past} (resp. α_{future}) does not solve t; and
- if $\delta[t] = j$ and the j-th algorithm is determined from C (that is $\sigma[j] \in \mathcal{A} \cap X_i$ or $\sigma[j]$ is the index of the first “past” or “future”, respectively), then the j-th algorithm defined by C solves t.

Note that if we take any valid schedule, we can project it w.r.t. a bag X_i and obtain a configuration $(\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma, \delta)$. Such a configuration will always be admissible and so we can restrict our attention to admissible configurations only.

To simplify the notation we let $\Gamma_i[C] = \infty$ if C is not an admissible configuration w.r.t. X_i.

Claim 12. *There are at most $2^\ell \cdot \ell^{|X_i|} \cdot |X_i|! \cdot m^2$ admissible configurations.*

Proof of Claim. First, there are at most m^2 possibilities for the algorithms α_{past} and α_{future}. Now, let $a, b \in \mathbb{N}$ be such that $a = |X_i \cap \mathcal{A}|$ and $b = |X_i \cap T|$. Note that $a + b = |X_i|$. There are $\binom{\ell}{a} \leq \ell^a$ possible position for the algorithms of X_i in σ and for each of these we have $|X_i|!$ possibilities for the specific algorithm to be at the given position. Then we have $2^{\ell-a}$ possibilities for the remaining entries of σ. Finally, there are ℓ^b functions from $X_i \cap T$ to $[\ell]$. Hence there are at most $m^2 \cdot \ell^a \cdot a! \cdot 2^{\ell-a} \cdot \ell^b \leq 2^\ell \cdot \ell^{|X_i|} \cdot |X_i|! \cdot m^2$ admissible configurations. □

Now for each X_i, we will compute a table Γ_i that contains an entry for each admissible configuration C such that $\Gamma_i[C] \in \mathbb{N}$ is the best cost, w.r.t. configuration C, of the already introduced tests restricted to the already introduced algorithms and the algorithm α_{future}.

Clearly, the minimum cost schedule of the instance gives rise to some admissible configuration C w.r.t. the root node X_r of the tree decomposition. Hence $\Gamma_r[C]$ contains the minimum cost of a schedule of length ℓ. In the remainder of the proof, we show how to update the respective nodes of tree decomposition depending on their children.

Claim 13. *If X_i is a leaf node, then Γ_i can be computed in $\mathcal{O}(|\Gamma_i|)$ time.*

Proof of Claim. Note that $X_i = \emptyset$ and that none of the algorithms has been introduced in any leave node. Let σ_{future} denote the function $\sigma_{future}[i] =$ "future" for all $i \in [\ell]$. Then the only admissible configuration looks like $(\emptyset, \alpha, \sigma_{future}, \emptyset)$, where $\alpha \in \mathcal{A}$. Moreover, since no tests or algorithms were introduced at that point, the cost of all of these configurations is zero. □

Claim 14. *If X_i is an introduce node for a test with the only child X_j, then Γ_i can be computed in $\mathcal{O}(|\Gamma_i|)$ time.*

Proof of Claim. Let t be the newly introduced test and let $C = (\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma, \delta)$ be an admissible configuration w.r.t. X_i. Note that since, X_j is a separator between the forgotten algorithms and t, it follows that α_{past} solves t and hence, because C is admissible, the index $\delta[t]$ of the algorithm that solve the test t has to be smaller or equal to the index of the first "past" in σ. Hence, it is simple to compute the cost c_t of t w.r.t. all already introduced algorithms and α_{future}. Moreover, this cost is unique and does not depend on forgotten algorithms other than α_{past}. Now, let $C' = (\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma, \delta')$ be the configuration w.r.t. X_j, where δ' is the restriction of δ missing the entry for t. Since $\Gamma_j[C']$ contains the best cost w.r.t. C' and the difference between C' and C and between X_j and X_i respectively is only the test t, it is easy to observe that $\Gamma_i[C] = \Gamma_j[C'] + c_t$. □

Claim 15. *If X_i is an introduce node for an algorithm with the only child X_j, then Γ_i can be computed in $\mathcal{O}(|\Gamma_i|)$ time.*

Proof of Claim. Let $C = (\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma, \delta)$ be an admissible configuration w.r.t. X_i and let α be the newly introduced algorithm. We compute $\Gamma_i[C]$ as follows. Let $\sigma' : [\ell] \to \{\text{"past"}, \text{"future"}\} \cup (\mathcal{A} \cap X_i$ be the function such that $\sigma'[k] = \sigma'[k]$ if $\sigma[k] \neq \alpha$ and $\sigma'[k] =$ "future" if $\sigma[k] = \alpha$.

We distinguish two cases depending on the position of α in σ.

First, $\sigma[k] = \alpha$ and there exists $k' \in \mathbb{N}$ such that $k' < k$ and $\sigma[k'] =$ "future". Let C' denote configuration $(\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma', \delta)$. Clearly, σ' is admissible w.r.t. X_j. Furthermore, all already forgotten tests are solved by α_{future}, which is before α in σ and hence their costs is already accounted for in $\Gamma_j[C']$. Therefore, we only need to account for the tests in X_i which are solved either by α or later. Let $k \in \mathbb{N}$ be such that $\sigma[k] = \alpha$. We let $\Gamma_i[C] = \Gamma_j[C'] + \sum_{t \in \{t' \mid \delta[t'] \leq k\}} \text{cost}(\alpha, t)$.

Second, $\sigma[k] = \alpha$ and there does not exist $k' \in \mathbb{N}$ such that $k' < k$ and $\sigma[k'] =$ "future". We let C' denote configuration $(\alpha_{\text{past}}, \alpha, \sigma', \delta)$. Clearly, σ' is admissible w.r.t. X_j. All already forgotten tests are solved latest by α and their costs is already accounted for in $\Gamma_j[C']$. However, we added new α_{future} in this case and we have to add the cost of running α_{future} for the tests in X_i that are solved after the first "future" in σ. Let $k \in \mathbb{N}$ be such that $\sigma[k] =$ "future" and for all $k' < k$ is $\sigma[k] \neq future$. We let $\Gamma_i[C] = \Gamma_j[C'] + \sum_{t \in \{t' \mid \delta[t'] \leq k\}} \text{cost}(\alpha_{\text{future}}, t)$. □

Claim 16. *If X_i is a forget node, which forgets a test t, with the only child X_j, then Γ_i can be computed in $\mathcal{O}(\ell|\Gamma_i|)$ time.*

Proof of Claim. Let $C = (\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma, \delta)$ be an admissible configuration w.r.t. X_i. Forgetting a test does not change the costs of introduced tests w.r.t. introduced algorithms. Hence, we only need to find a configuration w.r.t. X_j of the lowest cost that after removing t from δ results in C. For $k \in [\ell]$ let C_k be a configuration $(\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma, \delta_k)$ such that $\delta_k[t'] = \delta[t']$ for $t \in (X_i \cap T)$ and $\delta_k[t'] = k$. We let $\Gamma_i[C] = \min_{k \in [\ell]} \Gamma_j[C_k]$. □

Claim 17. *If X_i is a forget node, which forgets an algorithm α, with the only child X_j, then Γ_i can be computed in $\mathcal{O}((\ell+m)|\Gamma_i|)$ time.*

Proof of Claim. Let $C = (\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma, \delta)$ be an admissible configuration w.r.t. X_i. Clearly, when we forget an algorithm, the cost of schedule given by σ w.r.t already introduced algorithms and tests does not change. Hence, we just need to choose the best configuration of X_j that can result in C. Let k be the lowest index such that $\sigma[k] =$ "past". We distinguish two cases depending on whether $\alpha_{\text{past}} = \alpha$ or not.

First, if $\alpha_{\text{past}} = \alpha$, then for an already forgotten algorithm α' let us denote by $C_{\alpha'}$ the configuration $(\alpha', \alpha_{\text{future}}, \sigma', \delta)$ such that $\sigma'[k] = \alpha$ and for all $k' \neq k$ $\sigma'[k] = \sigma[k]$. In this case we let $\Gamma_i[C] = \min_{\alpha'} \Gamma_j[C_{\alpha'}]$.

If $\alpha_{\text{past}} \neq \alpha$, then for all $k' > k$ such that $\sigma[k'] =$ "past" we let $C_{k'}$ be the configuration $(\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma_{k'}, \delta)$ such that $\sigma'[k'] = \alpha$ and $\sigma'[k''] = \sigma[k'']$ for $k'' \neq k'$. We let $\Gamma_i[C] = \min_{k'} \Gamma_j[C_{k'}]$. □

Claim 18. *If X_i is a join node with children X_{j_1} and X_{j_2}, then Γ_i can be computed from Γ_{j_1} and Γ_{j_2} in $\mathcal{O}(2^\ell m|\Gamma_i|)$ time.*

Proof of Claim. Let $C = (\alpha_{\text{past}}, \alpha_{\text{future}}, \sigma, \delta)$ be an admissible configuration w.r.t. X_i. In this case, we need to go through all the possibilities how C can decompose into two admissible configurations $C_{j_1} = (\alpha^1_{\text{past}}, \alpha^1_{\text{future}}, \sigma^1, \delta^1)$ and $C_{j_2} = (\alpha^2_{\text{past}}, \alpha^2_{\text{future}}, \sigma^2, \delta^2)$ w.r.t. X_{j_1} and X_{j_2}, respectively. We first enumerate the necessary conditions for such two configurations.

1. $\delta = \delta^1 = \delta^2$,
2. σ, σ^1, and σ^2 agree on the position of the algorithms in X_i,
3. if $\sigma[k] =$ "future", then necessarily $\sigma^1[k] = \sigma^2[k] =$ "future",
4. if $\sigma[k] =$ "past" then either $\sigma^1[k] =$ "past" and $\sigma^2[k] =$ "future" or $\sigma^1[k] =$ "future" and $\sigma^2[k] =$ "past",
5. α_{past} has to be equal to either α^1_{past} or α^2_{past}, and
6. α^1_{future} is either α_{future} or α^2_{past} and α^2_{future} is either α_{future} or α^1_{past}.

It is straightforward to see that there are at most $2^\ell + 2 \cdot m$ such (C_{j_1}, C_{j_2}) pairs. Having such a pair and costs $\Gamma_{j_1}[C_{j_1}]$, $\Gamma_{j_2}[C_{j_2}]$, we show how to compute the cost of C if it rises from the combination of C_{j_1} and C_{j_2}. Afterwards, we just need to go through all the combinations and pick the one that gives the lowest possible cost.

First note that the sets of tests that are forgotten in X_{j_1} and X_{j_2} are disjoin,t and the cost of these tests is included in $\Gamma_{j_1}[C_{j_1}]$ and $\Gamma_{j_2}[C_{j_2}]$ respectively and this cost will not change after joining these to configurations. This is true since the only not introduced algorithms, in the respective bags, that will run on these tests are α^1_{future} and α^2_{future}, respectively, and their costs are already included in $\Gamma_{j_1}[C_{j_1}]$ and $\Gamma_{j_2}[C_{j_2}]$. Hence, if X_i does not contain any test, then the cost of combining these two configurations is precisely $\Gamma_{j_1}[C_{j_1}] + \Gamma_{j_2}[C_{j_2}]$. Similarly, for a test $t \in X_i$ and a forgotten algorithm other than α^1_{past} or α^2_{past} (which could be α^2_{future} or α^1_{future}, respectively) the cost of running this algorithm is already counted in precisely one of $\Gamma_{j_1}[C_{j_1}]$ and $\Gamma_{j_2}[C_{j_2}]$. Hence, the only thing that can be counted twice in $\Gamma_{j_1}[C_{j_1}] + \Gamma_{j_2}[C_{j_2}]$ is the cost of running a test in X_i w.r.t. algorithm in $X_i \cup \{\alpha_{\text{future}}, \alpha^1_{\text{future}}, \alpha^2_{\text{future}}\}$, which can be easily checked, computed from σ's and δ, and afterwards subtracted from $\Gamma_{j_1}[C_{j_1}] + \Gamma_{j_2}[C_{j_2}]$. Finally, if none of α^1_{future} and α^2_{future} is equal to α_{future}, we need to add to the final cost the cost of running α_{future} on tests that are in X_i and are solved at least at the position of the first "future" in σ or later (*e.g.*, $\delta[t] \geq k$, where k is the position of the first "future" in σ). This conclude the computation of the costs of combining two configurations from children in join node and by taking the minimum among all such combinations also the proof of the claim. □

To conclude, the last four claims show that it is possible to dynamically compute our records from the leaves of a nice tree decomposition to its root; once the records are known for the root, the algorithm has all the information it needs to output with the solution. □

Since, by Lemma 9 the length of a minimum cost schedule is at most $(2tw(G_{\mathcal{I}}) + 5) \cdot \log(n+m)$, by applying the above algorithm for each length between 1 and $(2tw(G_{\mathcal{I}}) + 5) \cdot \log(n+m)$ we get a runtime of $\mathcal{O}^*(4^{(2tw(G_{\mathcal{I}})+5)\cdot\log(n+m)} \cdot ((2tw(G_{\mathcal{I}}) + 5) \cdot \log(n+m))^{tw(G_{\mathcal{I}})} \cdot tw(G_{\mathcal{I}})!) = \mathcal{O}^*(4^{2tw(G_{\mathcal{I}})+5} \cdot (2tw(G_{\mathcal{I}}) + 5)^{tw} \cdot \log(n+m)^{tw(G_{\mathcal{I}})} \cdot tw(G_{\mathcal{I}})!)$. It is well known [20] that, for

a parameter k and input size N, a running time of the form $\mathcal{O}^*((\log(N))^k)$ is FPT. It follows that CPS[*failure treewidth*] is fixed-parameter tractable, and the proof of Theorem 7 is complete.

5.2 Using the Success Cover Number

The aim of this section is to establish the fixed-parameter tractability of CPS[*success cover number*], which can be viewed as a dual result to Corollary 8. The techniques used to obtain this result are entirely different from those used in the previous subsection; in particular, the proof is based on a significant extension of the ideas introduced in the proof of Proposition 1.

Theorem 19. CPS[*success cover number*] is in FPT.

Proof. Let $\mathcal{I}$ be an instance of CPS[cov_s]. Our first step is to compute a witness for the success cover number cov_s, *i.e.*, a set of algorithms $\mathcal{A}'$ and tests T' such that $|\mathcal{A}' \cup T'| = \mathrm{cov}_s$ and each pair in S has a non-empty intersection with $\mathcal{A}' \cup T'$; as discussed in Subsection 2, this can be done in polynomial time [8, Proposition 1]. Let $V = 2^{\mathcal{A}' \cup T'}$ be the set of all subsets of cov_s. We will construct a directed arc-weighted graph D with vertex set $V \cup \{x\}$, and with the property that each shortest path from $\emptyset$ to x precisely corresponds to a minimum-cost schedule for the input instance $\mathcal{I}$. Intuitively, reaching a vertex v in D which corresponds to a certain set of algorithms $\mathcal{A}_0$ and tests T_0 means that the schedule currently contains the algorithms in $\mathcal{A}_0$ plus an optimal choice of algorithms which can process the remaining tests in T_0; information about the ordering inside the schedule is not encoded by the vertex v itself, but rather by the path from $\emptyset$ to v.

In order to implement this idea, we will add the following arcs to D. To simplify the description, let $\mathcal{A}^*$ be an arbitrary subset of $\mathcal{A}'$ and T^* be an arbitrary subset of T'. First of all, for each $\mathcal{A}^*$ such that for every test $t \in T \setminus T'$ there is some $\alpha \in \mathcal{A}^*$ satisfying $(\alpha, t) \in S$, we add the arc $(\mathcal{A}^* \cup T', x)$ and assign it a weight of 0. This is done to indicate that $\mathcal{A}^* \cup T'$ corresponds to a valid schedule.

Second, for each $\mathcal{A}^*$ that is a proper subset of $\mathcal{A}'$, $\alpha_0 \in \mathcal{A}' \setminus \mathcal{A}^*$, and T^*, we add the arc e from $\mathcal{A}^* \cup T^*$ to $\mathcal{A}^* \cup \{\alpha_0\} \cup T^* \cup T_0$, where T_0 contains every test $t_0 \in T'$ such that $(\alpha_0, t_0) \in S$. In order to compute the weight of this arc e, we first compute the set T_e of all tests outside of T^* where α_0 will be queried (assuming α_0 is added to the schedule at this point); formally, $t \in T_e$ if $t \notin T^*$ and for each $\alpha' \in \mathcal{A}^*$ it holds that $(\alpha', t) \notin S$. For clarity, observe that $T_0 \subseteq T_e$. Now, we set the weight of e to $\sum_{t \in T_e} \mathrm{cost}(\alpha_0, t)$.

To add our third and final set of edges, we first pre-compute for each $T_\lambda \subseteq T' \setminus T^*$ an algorithm $\alpha_\lambda \in \mathcal{A} \setminus \mathcal{A}'$ such that:

1. for each $t_\lambda \notin T^*$, $(\alpha_\lambda, t_\lambda) \in S$ iff $t_\lambda \in T_\lambda$ (*i.e.*, α_λ successfully solves exactly T_λ), and
2. among all possible algorithms satisfying the above condition, α_λ achieves the *minimum cost* for all as-of-yet-unprocessed tests. Formally, α_λ minimizes the term $\mathrm{price}(\alpha_\lambda) = \big(\sum_{t \in (T' \setminus T^*)} \mathrm{cost}(\alpha_\lambda, t)\big) + \big(\sum_{t \notin T' : \forall \alpha \in \mathcal{A}^* : (\alpha, t) \notin S} \mathrm{cost}(\alpha_\lambda, t)\big)$.

Now, we add an arc e from each $\mathcal{A}^* \cup T^*$ to each $\mathcal{A}^* \cup T^* \cup T_\lambda$, where T_λ is defined as above and associated with the test α_λ. The weight of e is precisely the value $\mathrm{price}(\alpha_\lambda)$.

Note that since the graph D has $2^{\mathrm{cov}}_s + 1$ many vertices, a shortest path P from $\emptyset$ to x in D can be computed in time $2^{\mathcal{O}(\mathrm{cov}_s)}$. Moreover, it is easy to verify that D can be constructed from an instance $\mathcal{I}$ in time at most $2^{\mathcal{O}(\mathrm{cov}_s)} \cdot |\mathcal{I}|^2$. At this point, it remains to verify that a shortest $\emptyset$-x path P in D can be used to obtain a solution for $\mathcal{I}$.

Consider such a path P, and let Q be the schedule constructed iteratively as follows. At the beginning we set $Q = \emptyset$. We will follow the path P from $\emptyset$ to x and add an algorithm to Q whenever P traverses an arc with non-zero weight. In particular, whenever P uses an arc from some $\mathcal{A}^* \cup T^*$ to some $\mathcal{A}^* \cup \{\alpha_0\} \cup T^* \cup T_0$ where $\alpha_0 \in \mathcal{A}'$ (*i.e.*, an arc from our second group of created edges), we add α_0 to the end of Q. Similarly, whenever P uses an arc from some $\mathcal{A}^* \cup T^*$ to some $\mathcal{A}^* \cup T^* \cup T_\lambda$, we add the algorithm α_λ to the end of Q. It is easy to verify that at each point of our iterative construction of Q, the cost of an arc used by P is precisely the same as the processing cost the algorithm that was last added to Q spends on all tests which remain unsolved at this point. Moreover, by the construction of our first group of arcs, once P reaches x we end up with a valid schedule Q. It follows that Q is a valid schedule with cost equal to the weight of P.

To conclude the proof, among all valid schedules of minimum cost, let Q' be one such schedule with the minimum weight. It now suffices to show that there exists a path P' with the same weight as

the cost of Q'. To find this path P', we will be essentially be reversing the arguments made in the previous paragraph. Notably, we will find P' by choosing the i-th arc to follow based on the i-th algorithm in Q'. As our inductive assumption, we will moreover claim that:

1. if the i-th node we reach in P' is $\mathcal{A}^* \cup T^*$, then the first i algorithms in Q' contain $\mathcal{A}^*$ and that the algorithms in Q' solve precisely those tests in T' which lie in T^*; and
2. the weight of P' up to the i-th node is precisely equal to the cost of processing all tests by the first i algorithms in Q'.

This inductive assumption is clearly true at position $i = 0$, since P' starts at $\emptyset$. So, assume the inductive assumption holds at some position i, the vertex reached by P' after taking the i-th arc is some $\mathcal{A}^* \cup T^*$, and the $(i+1)$-st algorithm in Q' is some α_j.

If $\alpha_j \in \mathcal{A}'$, then we make our path P' follow the arc to $\mathcal{A}^* \cup \{\alpha_j\} \cup T^* \cup T_0$, where T_0 contains every test $t_0 \in T'$ such that $(\alpha_j, t_0) \in S$; such an arc can be found among the arcs created in our second group, the set T_0 is constructed in a way which preserves our first inductive assumption, and the cost of this arc will precisely match the cost of processing all remaining tests by α_j. Hence, both inductive assumptions will remain satisfied.

On the other hand, if $\alpha_j \notin \mathcal{A}'$, then due to the optimality of Q' we can assume that there exists some non-empty set $T_\lambda \subseteq T' \setminus T^*$ of tests solved by α_j, *i.e.*, $t \in T_\lambda$ iff $(\alpha_j, t) \in S$. To be more precise, observe that an α_j outside of $\mathcal{A}'$ can only successfully process tests in T', and if such α_j does not succeed on any as-of-yet remaining test we may simply remove α_j from Q', contradicting our assumption about the (length) minimality of Q'. Moreover, among *all* algorithms which succeed precisely on T_0, we note that α_j must run with the minimum cost for all test instances that remain unsolved as of this point—in other words, α_j must be an algorithm with minimum price, as defined when constructing our third set of edges. Indeed, if this were not the case, we could replace α_j with the algorithm α_λ pre-computed for T_λ, leading to a schedule Q' of strictly smaller cost. But since α_j is an algorithm with minimum price, there must exist an arc from $\mathcal{A}^* \cup T^*$ to $\mathcal{A}^* \cup T^* \cup T_\lambda$ of cost precisely price(α_j), and by having P' use this arc we ensure that both inductive assumptions remain valid (the first assumption follows from the definition of T_λ, and the second follows from the optimality of the price of α_j).

To summarize, we have shown that for each valid optimal schedule Q' as defined above, there is a corresponding $\emptyset$-x path P' in our graph D, and that for each $\emptyset$-x path P in D there is a corresponding valid schedule. These two facts together establish the correctness of our algorithm. □

6 Conclusion

We studied the parameterized complexity of the CASCADING PORTFOLIO SCHEDULING problem under various parameters. We identified several settings where the NP-hardness of the problem can be circumvented via exact fixed-parameter algorithms, including cases where (i) the algorithms have a small failure degree, (ii) the tests have a small failure degree, (iii) the evaluation matrix has a small failure cover, and (iv) the evaluation matrix has a small success cover. The first three cases can be seen as settings in which most algorithms succeed on most of the tests, whereas case (iv) can be seen as a setting where most algorithms fail.

We have complemented our algorithmic results with hardness results which allowed us to draw a detailed complexity landscape of the problem. We would like to point out that all our hardness results hold even when all costs are unit costs. This finding is significant, as it reveals that the complexity of the problem mainly depends on the success relation and not on the cost mapping.

For future work, it would be interesting to extend our study to the more complex setting where up to p algorithms from the portfolio can be run in parallel. Here, the number p could be seen as a natural additional parameter.

References

[1] H. L. Bodlaender. Discovering treewidth. In *Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'05)*, volume 3381 of *Lecture Notes in Computer Science*, pages 1–16. Springer Verlag, 2005.

[2] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*,

45(2):317–378, 2016.

[3] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[4] M. Cygan, F. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[5] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[6] Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

[7] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer Verlag, Berlin, 2006.

[8] Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. Parameterized algorithms for the matrix completion problem. In *Proceeding of ICML, the Thirty-fifth International Conference on Machine Learning, Stockholm, July 10–15, 2018*, pages 1642–1651. JMLR.org, 2018. ISSN: 1938-7228.

[9] Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artificial Intelligence*, 174(1):105–132, 2010.

[10] Georg Gottlob and Stefan Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction, and database problems. *The Computer Journal*, 51(3):303–325, 2006. Survey paper.

[11] Holger H. Hoos, Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider. Portfolio-based algorithm selection for circuit QBFs. In John N. Hooker, editor, *Proceedings of CP 2018, the 24rd International Conference on Principles and Practice of Constraint Programming*, volume 11008 of *Lecture Notes in Computer Science*, pages 195–209. Springer Verlag, 2018.

[12] Shinji Ito, Daisuke Hatano, Hanna Sumita, Akihiro Yabe, Takuro Fukunaga, Naonori Kakimura, and Ken-ichi Kawarabayashi. Regret bounds for online portfolio selection with a cardinality constraint. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 10611–10620, 2018.

[13] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, pages 1–47, 2018.

[14] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.

[15] Marius Lindauer, Holger Hoos, Frank Hutter, and Kevin Leyton-Brown. Selection and configuration of parallel portfolios. In *Handbook of Parallel Constraint Reasoning.*, pages 583–615. 2018.

[16] Marius Lindauer, Frank Hutter, Holger H. Hoos, and Torsten Schaub. Autofolio: An automatically configured algorithm selector (extended abstract). In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 5025–5029. ijcai.org, 2017.

[17] Haipeng Luo, Chen-Yu Wei, and Kai Zheng. Efficient online portfolio with logarithmic regret. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 8245–8255, 2018.

[18] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford, 2006.

[19] Luca Pulina and Armando Tacchella. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, 14(1):80–116, 2009.

[20] V. Raman, S. Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for undirected feedback vertex set. In *Proceedings of the 13th International Symposium on Algorithms and Computation*, volume 2518 of *Lecture Notes in Computer Science*, pages 241–248. Springer, 2002.

[21] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[22] Mattia Rizzini, Chris Fawcett, Mauro Vallati, Alfonso Emilio Gerevini, and Holger H. Hoos. Static and dynamic portfolio methods for optimal planning: An empirical analysis. *International Journal on Artificial Intelligence Tools*, 26(1):1–27, 2017.

[23] Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

[24] Olivier Roussel. Description of ppfolio 2012. In et al. A. Balint, editor, *Proceedings of SAT Challenge 2012*, page 47. University of Helsinki, 2012.

[25] Matthew Streeter. Approximation algorithms for cascading prediction models. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 4759–4767. JMLR.org, 2018.