The solution space of sorting with recurring comparison faults

Downloaded from: https://research.chalmers.se, 2026-09-05 05:53 UTC

Citation for the original published paper (version of record):

Damaschke, P. (2016). The solution space of sorting with recurring comparison faults. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9843 LNCS: 397-408 . http://dx.doi.org/10.1007/978-3-319-44543-4_31

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

Theory Comput Syst (2018) 62:1427–1442
DOI 10.1007/s00224-017-9807-4

The Solution Space of Sorting with Recurring Comparison Faults

Peter Damaschke[1]

Published online: 31 August 2017
© The Author(s) 2017. This article is an open access publication

Abstract Suppose that n elements shall be sorted by comparisons, but some subset of at most k pairs systematically returns false comparison results. This subset is unknown, but the number k is known in advance. Using a connection to feedback arc sets in tournaments (FAST), we characterize the solution space of sorting with recurring comparison faults by a FAST enumeration, which represents all information about the order that can be obtained by doing all $\binom{n}{2}$ comparisons. Some optimal parameterized enumeration algorithm for FAST also works for the more general chordal graphs, and this fact contributes to the efficiency of our representation. Next we compute the solution space more efficiently, by fault-tolerant versions of Treesort and Quicksort. We need $O(n \log n + kn + k^2 \log n)$ comparisons and $O(n \log n + kn + k^2 \log n + kF(k^2, k))$ time, where $F(n, k)$ is any parameterized time bound for finding a FAST with at most k arcs. Thus, for rare faults the complexity is close to optimal. We also propose directions of further research, revolving around decision diagrams for sorting with recurring faults.

Keywords Sorting · Faulty input · Feedback arc set · Enumeration · Fixed-parameter tractability

This article is part of the Topical Collection on *Special Issue on Combinatorial Algorithms*

✉ Peter Damaschke
ptr@chalmers.se

[1] Department of Computer Science and Engineering, Chalmers University, 41296 Göteborg, Sweden

1 Introduction

In the model of recurring faults in computations, as introduced in [11], operations on certain items yield false results even when they are repeated. As opposed to transient or probabilistic failures, this model accounts for systematic errors. One of the problems investigated in [11] is to sort a set of n elements by comparisons, under the assumption A_k that at most k pairs return false comparison results. Recurring comparison faults can origin from software bugs or, most notably, from unreliable floating-point operations in geometric computations [10]. One can also think of applications where the elements are real entities rather than data items in computer memory. For instance, archeological finds or historical events may be brought into chronological order by pairwise comparisons, say by comparing style characteristics or by causal dependencies, respectively, but for a few pairs the comparison criteria may be misleading.

It is impossible to verify assumption A_k from the comparison results only, because an adversary may choose any order and just give consistent answers. The best we can do is to determine all orders compatible with A_k, and then we know: *If A_k holds true, then these are the possible orders.* Only if no compatible order exists, we recognize that A_k is false. Hence the problem belongs to the category of promise problems: We must know in advance that comparisons are reliable, subject to a certain "small" number of at most k false pairs, where k is known in advance.

In [11], quality measures for alleged sorted sequences are defined and related to each other. This is done from the approximation point of view, thus asking: How much does an order obtained by doing *all comparisons* and some postprocessing differ from the unknown sorted order? (See also some earlier related work in [10].) What is not considered is the full solution space obtained from the comparisons, and the number of comparisons actually needed. A fault-tolerant search algorithm for the minimum element is provided in [11]. It returns an element of rank $O(k)$ by using $O(\sqrt{kn})$ comparisons and time. Here we aim at similar results for the sorting problem. We separate the number of comparisons and auxiliary computations, as comparisons may be more expensive, depending on the nature of elements to compare.

Our Contributions We answer two different questions: 1. What can we learn at all about an unknown order by faulty comparisons? 2. How can we efficiently extract the entire information (that is asked for in point 1.)? Specifically, how can we infer all comparison results by doing only a minority of them, ideally in a time close to $O(n \log n)$?

We give an overview of our contributions. (Here it is hard to summarize the contents without already being technical; for definitions see Section 2.)

Since arcs (u, v) from true comparison results $u < v$ cannot form directed cycles, sorting with k faults is closely related to minimal feedback arc sets (MFAS) of size k in directed graphs. Starting from a version of the previously known "reversal lemma" for MFAS, we enumerate in $O(3^k k(n + m))$ time all MFAS with at most k arcs in a directed graph of n vertices and m edges whose underlying undirected graph is chordal. This extends an early algorithm [13] for finding smallest MFAS in

tournaments, also called FAST. While it is known that a single minimum FAST can be computed faster, base 3 in the time bound is necessary (hence optimal) to generate an explicit enumeration, for simple reasons. Next we show that, in a precise sense, the MFAS enumeration characterizes the solution space, i.e., the set of orders compatible with all comparisons. While there can exist $n^{O(k)}$ such orders, it suffices to know at most 3^k MFAS, as all other compatible orders can be obtained from them by simple sequences of transpositions. Using the previous results we show that the MFAS that describe all compatible orders can be enumerated in $O(3^k k^2 n)$ time, once a single compatible order is known. Finally we give efficient algorithms that actually find a compatible order and all information needed to reconstruct the solution space. One building block is a procedure to insert another vertex in an existing order with a minimum number of backward arcs. This leads to fault-tolerant sorting algorithms based on Treesort and Quicksort. They essentially need $O(n \log n)$ comparisons for any fixed k, which is optimal in a sense. The time is larger by just some "FPT term" in the parameter k. These are the first subquadratic algorithms for sorting with recurring comparison faults. We also propose further research on succinct representations of the solution space using decision diagrams.

Other Related Literature As much work has been done on fault-tolerant searching and sorting (see the survey [5]), it is important to pay attention to similarities. First of all, different fault models have been considered. Liar models are deterministic fault models allowing some maximum number of false answers, like the model considered here. But the difference is that they count repeated false answers, hence the searcher may reconstruct every true answers simply by repetition and majority vote. Sorting in a model where some elements can be corrupted (but comparisons are correct) is considered in [8]. There the goal is to sort the uncorrupted elements. Sorting under probabilistic errors is studied in [3, 4]. Some steps of our insertion procedure resemble some of their lemmas, as well as arguments from the kernelization of FAST [1]. Sorting with faults was also studied in [10], however, in their model an algorithm may use expensive and cheap comparisons, where the cheap ones may err for elements whose ranks differ by at most some constant. Then, bounds on the number of inversions for several standard sorting algorithms are proved. Enumeration problems find attention in various fields (see, e.g. [2]). The number of comparisons needed to decide certain properties of partial orders has been studied in [7].

2 Preliminaries

A computational problem is fixed-parameter tractable (FPT) if instances with input size n and an additional input parameter k can be solved in $f(k) \cdot n^{O(1)}$ time, where f is some computable function.

In directed (undirected) graphs we refer to pairs of vertices as arcs (edges). As usual, n and m denotes the number of vertices and arcs (edges) of a graph. We use the terms *vertex* and *element* interchangeably.

In the following we introduce quite a number of special definitions, however we believe that they are necessary for formal clarity.

To keep order-theoretic terminology simple we consider ordered sets to be ascending from "left" to "right". Let V be the set of the n elements to be sorted. Any set of comparisons performed on pairs of elements naturally defines a directed graph $D = (V, A)$, where every arc (u, v) indicates a comparison that claimed $u < v$. We call $D = (V, A)$ a *comparison graph*. We call the arc (u, v) *true* if actually $u < v$ holds, and *false* if actually $v < u$ holds. With respect to an order σ of V, an arc is *forward (backward)* if it points to the right (left). We denote the set of backward arcs $B(\sigma)$. The *length* of an arc (u, v) is the absolute difference of the positions of u and v in σ. Let k be a fixed integer. Provided that at most k comparisons are false, clearly, an order σ is a candidate for the correctly sorted sequence if and only if $|B(\sigma)| \leq k$. As in [11], we call such an order σ *compatible*.

A *transposition* flips the positions of two neighbored vertices u, v in an order. It turns the arc (u, v), if there is one, from forward to backward or vice versa, while all other arcs are not affected. For two orders π and σ of the same set, an *inversion* is a pair of elements u, v such that u is to the left of v in π but v is to the left of u in σ. The *Kemeny distance* $d(\pi, \sigma)$ is the number of inversions. Starting from π, consider any sequence of transpositions with the property that each transposition removes an inversion. As a simple fact, every maximal sequence of this kind has length $d(\pi, \sigma)$ and ends in σ.

A directed graph $D = (V, A)$ is *acyclic* if it has no directed cycles. As is well known, a directed graph is acyclic if and only if it admits an order without backward arcs, called a *topological order*, moreover, one can construct a topological order or output a directed cycle in $O(n + m)$ time. For arbitrary directed graphs $D = (V, A)$ we call an order σ of V a *minimal backward* order if no other order τ with $B(\tau) \subset B(\sigma)$ exists. A *minimum backward* order has the smallest possible number of backward arcs. Hence, in acyclic graphs, minimum and minimal backward orders and topological orders are the same.

A *feedback arc set (FAS)* of a directed graph is a subset of arcs whose removal makes the graph acyclic, and a *minimal FAS (MFAS)* is a FAS such that no proper subset of it is a FAS, too. A *tournament* is a complete directed graph $D = (V, A)$. A (directed) *triangle* is a (directed) cycle of three vertices. The FAST problem requires to find a minimum FAS in a tournament. Let $F(n, k)$ be a time bound of an FPT algorithm for FAST, for graphs with $O(n)$ vertices and solution size k. Note that $F(n, k)$ is well-defined for any FPT algorithm: Since the dependency of the time bound on n is polynomial, a constant factor in n only affects the constant factor in $F(n, k)$. We will give time bounds in terms of $F(n, k)$; here it is not the intention to hide the exponential part, but to give the bounds in a generic way, independently of the current state of FAST. Specifically one may plug in $F(n, k) = 2^{O(\sqrt{k})} n^{O(1)}$ from [6, 9].

The undirected *underlying graph* of a directed graph is obtained by ignoring the orientations of arcs. An undirected graph is *chordal* if every cycle C is a triangle or has a *chord*, that is, an edge joining two non-consecutive vertices in C. Every chordal graph has a *perfect elimination order (PEO)*, defined by the following property: If u is the first of u, v, w in the order, and uv and uw are edges, then vw is an edge, too. Given a chordal graph, a PEO can be constructed in $O(n + m)$ time [14].

3 Characterizing and Enumerating MFAS

Before discussing our main topic, sorting with recurring faults, we give a few graph-theoretic results that will play some role. The "reversal lemma" states that reversing the arcs of an MFAS makes a directed graph acyclic. It was already discovered several times in the 1960s and also used in [13]. The following extended version also considers the corresponding orders.

Lemma 1 *An arc set $F \subseteq A$ is an MFAS in a directed graph $D = (V, A)$, if and only if $F = B(\sigma)$ for some minimal backward order σ. Moreover, the possible σ are exactly the topological orders of $(V, A \setminus F)$.*

Proof For any order σ, trivially, $B(\sigma)$ is a FAS. Let F be any FAS. Then $(V, A \setminus F)$ is acyclic. We take any topological order σ and re-insert the arcs of F. Clearly, $B(\sigma) \subseteq F$. If F is an MFAS then, since $B(\sigma)$ is a FAS, it also follows $B(\sigma) = F$. Now assume that σ is not minimal backward. Then there exists another σ' with $B(\sigma') \subset B(\sigma)$. But $F' := B(\sigma')$ is also a FAS, and $F' \subset F$ contradicts the minimality of F. Thus, every topological order of $(V, A \setminus F)$ is minimal backward.

Conversely, let σ be any minimal backward order, and $F := B(\sigma)$. Then F is a FAS. Assume that a smaller FAS $F' \subset F$ exists. As we saw above, there exists a topological order σ' such that $B(\sigma') \subseteq F' \subset F = B(\sigma)$, which contradicts the assumed backward minimality of σ. □

Lemma 2 *A directed graph with an underlying chordal graph is acyclic if and only if it has no directed triangle. Furthermore, we can confirm that the graph is acyclic or find a triangle in $O(n + m)$ time.*

Proof We run a standard $O(n+m)$ time algorithm that constructs a topological order or outputs a directed cycle. If the graph is acyclic, trivially it has no directed triangle. If we get a directed cycle C, represented as a doubly linked circular list, it remains to find a directed triangle in $O(n + m)$ time. To this end we construct in $O(n + m)$ time a PEO of the underlying chordal graph and mark the vertices of C therein. We scan the PEO from left to right until we find the first vertex $u \in C$. Let v and w be its neighbors in C (in the circular list). Then u, v, w form a triangle, due to the PEO. If this triangle is directed, we can stop. If not, then we update C by removing u and its two incident arcs, and inserting the arc (v, w) or (w, v) instead. The shortened cycle is still directed, and the update is done in $O(1)$ time. We keep on scanning the PEO until the next vertex of C is found. Since the cycle is shortened each time and remains directed, eventually we get a directed triangle. □

We are ready to state a result on MFAS enumeration when the underlying graph is chordal.

Theorem 1 *In a directed graph with chordal underlying graph, at most 3^k MFAS of at most k arcs exist, and they can be enumerated in $O(3^k k(n + m))$ time.*

Proof We pick any directed triangle T and branch on it. That means, we generate at most three sub-instances of the problem as follows: In every branch we choose one arc of T, reverse it and mark it. Marked arcs are not reversed again in later steps (dealing with other triangles). If all three arcs in T were already marked, then the sub-instance is discarded. Each of the, at most 3^k, paths of branching steps is followed until k steps are done or the obtained directed graph is free of directed triangles. We collect the latter graphs. By Lemma 2, each of them is acyclic, hence the reversed arcs form a FAS. Eventually we throw out all FAS that are not MFAS or are duplicates of other MFAS.

For correctness it remains to show that every MFAS F with at most k arcs is found in this collection. We use Lemma 1 and fix an order σ where $F = B(\sigma)$. We follow a path of reversals where only arcs of F get reversed. As long as the obtained graph is not acyclic, by Lemma 2, it retains a directed triangle. The algorithm picks some; let us call it T. Clearly, some of the three arcs in T is still backward in σ, thus the arc is in F and not yet reversed and marked, and one of the branches reverses just this arc. As soon as the obtained graph is acyclic, the graph without the reversed arcs is acyclic, too, but since F is an MFAS, it follows that all arcs of F have already been reversed. These two cases show that our path never gets stuck with a proper subset of F reversed.

We have $O(3^k)$ branching steps, and the main work in each of them is to find a directed triangle. By Lemma 2 this us done in $O(n+m)$ time. Every FAS F not being an MFAS is detected easily: For every arc e we check whether $F \setminus \{e\}$ is still a FAS, in $O(n+m)$ time. This costs $O(3^k k(n+m))$ time for all collected FAS. Duplicates are recognized by bucketsorting. □

We conclude this section with a few remarks. One could also make the enumeration repetition-free by sorting the edges in each triangle and marking the reversed arc and the preceding arcs, but we remove duplicates anyhow, and the base in the 3^k factor is optimal, even for tournaments. To see this, take for instance k disjoint directed triangles, arrange them in an order, and insert all possible forward edges between vertices of different triangles. Then each of the 3^k selections of one arc from each triangle is an MFAS. By this 3^k lower bound, none of the faster algorithms that compute a minimum FAS can be turned into a faster algorithm that enumerates all MFAS as an explicit list.

4 MFAS and the Solution Space of Faulty Sorting

In this section we characterize the family of all orders of a set being compatible with a comparison graph, by virtue of an MFAS enumeration and additional transpositions.

Lemma 3 *An order σ of the vertex set of a directed tournament $D = (V, A)$ is minimal backward if and only if no backward arc has length 1.*

Proof One direction is trivial: If some backward arc has length 1, then a transposition makes it a forward arc, hence σ is not minimal backward.

Conversely, assume for contradiction that no backward arc in σ has length 1, but there exists an order τ with $B(\tau) \subset B(\sigma)$. Consider an arc $(u, v) \in B(\sigma) \setminus B(\tau)$ that has minimum length in σ, among all arcs in this set difference. Since this length is not 1, there exists a vertex $w \in V$ such that v, w, u appear in this order in σ. Clearly, u appears before v in τ, hence w swaps its position relative to u or v or both. We look at the conceivable cases:

Assume that w appears before v in τ; the other case is symmetric. If $(v, w) \in A$ then $(v, w) \notin B(\sigma)$ and $(v, w) \in B(\tau)$, which contradicts the choice of τ. If $(w, v) \in A$ then $(w, v) \in B(\sigma)$ and $(w, v) \notin B(\tau)$, but since (w, v) is shorter than (u, v), this contradicts the choice of (u, v). □

Theorem 2 *For a tournament $D = (V, A)$ and an integer k, every compatible order can be obtained from a compatible, minimal backward order by a sequence of transpositions, each turning a (current) forward arc into a backward arc.*

Proof Consider any compatible order σ. If σ is not minimal backward, then, by Lemma 3, it has a backward arc of length 1. A transposition at this place removes exactly this arc from $B(\sigma)$. By an inductive argument, a sequence of such transpositions ends in some minimal backward order. (We remark that this final order is not unique.) Trivially, this order is compatible, too. The assertion follows by reversing the sequence of transpositions. □

Suppose that we have done all $\binom{n}{2}$ comparisons, that is, the comparison graph is a tournament. Then, the results provide a simple implicit description of all compatible orders, which is also practical for rare faults, that is, for small numbers k:

Enumerate all MFAS with at most k arcs in the comparison graph, in $O^*(3^k)$ time (as in Theorem 1). For any solution, reverse the arcs in the MFAS, and output the resulting order. By Lemma 1 it is a compatible, minimal backward order If the number of backward arcs is $b < k$, all other compatible orders are obtained by up to $k - b$ transpositions that preserve the backward arcs; subject to this condition, the transpositions are arbitrary. Equivalently, these orders have Kemeny distance at most $k - b$ from the minimal backward orders. We emphasize that Theorem 2 is not an isolated observation but an integral part of the characterization of the solution space. It implies that algorithms for fault-tolerant sorting need to care about minimal backward orders only.

We have not illustrated the concepts and theorems by small single examples of comparison graphs, since they would not be very "respresentative". Instead we have chosen to discuss some classes of examples whose structure may be of particular interest (see below).

If no backward arcs exist ($b = 0$), then the compatible orders are exactly those which can be turned into the correct order by k steps of Bubblesort. Next consider an order with a single backward arc ($b = 1$):

Example: Single Backward Arc Let (v, u) be the only backward arc, where u is d positions to the left of v. One MFAS consists of only (v, u), and the corresponding

minimal backward order is the given order. Any MFAS not containing (v, u) must, for each of the $d-1$ vertices w between u and v, contain (u, w) or (w, v). We get d different MFAS, each with $d-1$ arcs, as follows: Divide the segment between u and v into a prefix and a suffix. (One of them can be empty.) Take (u, w) for all w in the prefix, and take (w, v) for all w in the suffix. The corresponding minimal backward order is obtained by swapping u (v) with all vertices of the prefix (suffix), and finally swapping u and v. (Indeed, all backward arcs get lengths larger than 1.) Since all directed triangles are of the form (v, u, w), these $d+1$ MFAS are all possible MFAS, even if $k > d-1$ is permitted. Reversing the backward arcs in any minimal backward order yields a topological order. Below we refer to the $d+1$ alternative MFAS as "branches".

Example: Non-Overlapping Backward Arcs Extending the previous example, consider an order with several backward arcs (v_1, u_1), (v_2, u_2), (v_3, u_3), etc., such that every v_i is to the left of u_{i+1}. That is, the segments spanned by the backward arcs are pairwise disjoint. Let d_i denote the length of (v_i, u_i). Since the backward arcs do not "interfere", the above branches apply independently to the backward arcs, which yields $\prod_i (d_i + 1)$ different MFAS. The case of non-overlapping backward arcs may be typical, since comparison faults may be more likely for elements being close in the correct order.

For instance, these two backward arcs of lengths $d_1 = 2$ and $d_2 = 3$

$$\begin{array}{cccccccccc} u_1 & \longleftarrow & v_1 & u_2 & \longleftarrow & \cdots & -v_2 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}$$

lead to $(2+1) \cdot (3+1) = 12$ MFAS and the following orders:

$$\begin{array}{lll} (1,2,3,4,5,6,7) & (2,3,1,4,5,6,7) & (3,1,2,4,5,6,7) \\ (1,2,3,5,6,7,4) & (2,3,1,5,6,7,4) & (3,1,2,5,6,7,4) \\ (1,2,3,5,7,4,6) & (2,3,1,5,7,4,6) & (3,1,2,5,7,4,6) \\ (1,2,3,7,4,5,6) & (2,3,1,7,4,5,6) & (3,1,2,7,4,5,6) \end{array}$$

5 A Certificate for Sorting with Recurring Faults

Sorting without faults (the case $k = 0$) needs only $O(n \log n)$ comparisons. Therefore the next natural question is whether we can construct the solution space for faulty sorting without doing all $O(n^2)$ comparisons, at least when k is small. We will provide an affirmative answer. First we give a short intuitive explanation of our approach, however the reader may skip the next paragraph and go straight to the technical details.

If we just applied any standard $O(n \log n)$-time sorting algorithm and temporarily believed all comparison results, we would get some order being consistent with them, but in general we would not recognize potential faults. Now the main idea is to spot errors by additionally comparing all pairs of elements at small distances (depending on k) in a given ordering. Actually, we will see that comparing all elements at distances at most $2k+1$ is sufficient to certify that a given order is compatible with

A_k, and then all pairs of elements at larger distances are already in the correct order. The final algorithms, however, will mimic standard sorting algorithms but will do the additional comparisons of nearby elements already during the construction of compatible orders.

Definition 1 Given an order σ of the vertex set V, let $D(\sigma)$ be the comparison graph consisting of V and all arcs of length at most $2k+1$ in σ.

Note that $D(\sigma)$ has $O(kn)$ arcs, and its underlying graph is chordal. (The latter is seen, e.g., as follows: Every cycle has a leftmost vertex in σ, and its two neighbors in the cycle are within distance $2k+1$ of each other in σ, hence there is an edge between them.)

Consider the case when, after running an $O(n \log n)$-time sorting algorithm, we get an order σ such that $D(\sigma)$ happens to have at most k backward arcs. In this case we are in a good position: The next theorem says that further comparisons do not add more information, thus the instance of the sorting problem would be solved after $O(n \log n + kn)$ comparisons, namely $O(n \log n)$ and $O(kn)$ comparisons from sorting and from $D(\sigma)$, respectively.

Theorem 3 *Consider an ordered comparison graph that contains all arcs of length at most $2k+1$, where at most k of them are backward arcs. Let u and v be any two vertices such that v appears more than $2k+1$ positions to the right of u. Then we can safely conclude $u \prec v$. (In other words, the arcs not being in $D(\sigma)$ are forward in all compatible orderings.)*

Proof We use induction on the distance d between u and v in the order. Suppose that the assertion holds for all distances between $2k+1$ and d. Let w be any of the $d-1$ vertices between u and v. We have either (1) $u \prec w$ by the induction hypothesis, or (2) (u, w) is a forward arc, or (3) (w, u) is a backward arc. Similarly, we have either (1) $w \prec v$ by the induction hypothesis, or (2) (w, v) is a forward arc, or (3) (v, w) is a backward arc.

Since at most k backward arcs exist, for at least $d-1-k$ of the vertices w, only cases (1) and (2) apply, with respect to both u and v. Since at most k arcs are false, for at least $d-1-2k \geq 1$ of the vertices w, we have both $u \prec w$ and $w \prec v$, where each of the two inequalities holds either by the induction hypothesis or since the forward arc, (u, w) or (w, v), is true. Note that we do not know which forward arcs are true, yet we can infer the existence of a vertex w with $u \prec w \prec v$. This concludes the induction step and the proof. □

By Theorem 3, a graph $D(\sigma)$ with at most k backward arcs is a certificate that all other arcs are forward. Thus, the solution space description from Section 4 can be based on $D(\sigma)$, as we know the directions of all other arcs without actually testing them. Since $D(\sigma)$ has $m = O(kn)$ edges and is chordal, by Theorem 1 we can enumerate its MFAS already in $O(3^k k^2 n)$ time, which is $O(n)$ for any fixed k.

However, of course, we cannot expect in general to be lucky and obtain some $D(\sigma)$ with at most k backward arcs already in one pass of a usual sorting algorithm.

The following sections deal with the actual construction of an order that satisfies the condition in Theorem 3. We conclude this section with another structural property that will be needed.

Definition 2 Consider a tournament and an order of its vertices. We partition it into *components* with the following properties: every component is a consecutive set of vertices; every backward arc is within a component; and for every position between two vertices in a component there exists a backward arc from a vertex on the right side to a vertex on the left side of this position. A *trivial component* has only one vertex, and a *nontrivial component* has more than one vertex.

From the definition it is evident that the components are uniquely determined. We index them from left to right by C_1, C_2, C_3, and so on. Let b_i denote the minimum number of backward arcs in an optimal order of C_i, and let $b := \sum_i b_i$. We define the following routine:

Procedure MB In every nontrivial component C_i, compute a minimum FAS. Due to Lemma 1, topological sorting then yields a minimal backward order of C_i. Rearrange the vertices within each C_i accordingly, but keep the order of the components.

Lemma 4 *The order obtained from MB has exactly b backward arcs, which is optimal.*

Proof The minimal backward order of every C_i has b_i backward arcs. Since we keep the order of components, and there exist no backward arcs between components, it is evident that exactly b backward arcs exist. To show optimality, consider any order of the whole set. The order induced on every C_i still has at least b_i backward arcs, since b_i is optimal in C_i. Since the components are disjoint, no backward arcs are counted twice. It follows that at least b backward arcs are needed. □

6 Insertion in a Compatible Minimum Backward Order

Now we want to efficiently construct an order of the vertex set V with at most k backward arcs.

Suppose that we have already found an order σ of a subset $U \subset V$, such that $D(\sigma)$ exhibits at most k backward arcs. Due to Theorem 3 this also implies that all longer arcs are forward. We can further suppose that the number of backward arcs in σ, or equivalently, in every component, is minimized (see Lemma 4). Let us store the sequence σ in an array indexed with consecutive integers. The next goal is to insert another vertex $v \notin U$ and to find an order τ of $U \cup \{v\}$ that still enjoys the same properties. Such an order must exist, if at most k comparison faults are present, but it is not obvious how to get τ efficiently from σ. We begin with a transitivity lemma and then establish a fault-tolerant binary search that runs, so to speak, on an almost sorted set blurred by comparison faults.

Lemma 5 *Suppose that u' stands to the left of u, at a distance larger than $2k+1$. If (u, v) is true, then $u' < v$. A similar assertion holds in the symmetric case.*

Proof By the assumed distance and Theorem 3, we have $u' < u$. Since (u, v) is true, we also have $u < v$, hence $u' < v$. □

Lemma 6 *We can find two elements ℓ and r with distance $O(k)$ in σ and with $\ell < v < r$, by using $O(k \log n)$ comparisons of elements of U with v, in $O(k \log n)$ time.*

Proof Let us append dummy vertices to σ: one at the left end which is smaller than all real elements, and one at the right end which is larger than all real elements. Initially let ℓ and r be these dummy elements, hence $\ell < v < r$ is true. To "query a vertex" means to compare it to v. Since σ is stored as an array, we have access to the indices and can find the center of an interval in $O(1)$ time.

We query consecutive vertices u around the center of the interval $[\ell, r]$, until $k+1$ of them give the same answer, say $u < v$. Clearly, this happens after at most $2k+1$ comparisons. Since at most k comparisons are false, we know that $u < v$ is true for some queried vertex u, but we cannot say which. However, Lemma 5 ensures $u' < v$ for all u' more than $2k+1$ positions to the left of u. Thus it is safe to update ℓ to the vertex at distance $2k+2$ to the left of the leftmost queried vertex. Similarly we proceed with r in the symmetric case. Thus, the property $\ell < v < r$ is preserved. In each step we halve the interval $[\ell, r]$ and add an offset of $O(k)$. Clearly, after $O(\log n)$ such steps with $O(k \log n)$ comparisons, the length of $[\ell, r]$ is reduced down to $O(k)$. □

This simple procedure may be viewed as binary search with error correction by majority vote. Next we finalize the procedure. Recall the FAST time bound $F(n, k)$ (which is subexponential in k) from Section 2, recall the notion of components from Definition 2, and note that a component has $O(k^2)$ vertices, since at most k backward arcs exist, all of length $O(k)$.

Lemma 7 *Given an order σ of U such that $D(\sigma)$ has a minimum number of backwards arcs, bounded by k, we can get an order τ of $U \cup \{v\}$ with the same properties, by $O(k \log n)$ comparisons in $O(k \log n + F(k^2, k) + n)$ time.*

Proof After running the procedure in Lemma 6 we insert v anywhere in $[\ell, r]$ and denote by σ' the resulting order of $U \cup \{v\}$. By Theorem 3, ℓ is larger than all vertices to the left, and r is smaller than all vertices to the right, with the exception of at most $2k+1$ vertices next to ℓ and r, respectively. From Lemma 6 we have $|[\ell, r]| = O(k)$ and $\ell < v < r$. Thus v is only involved in backward arcs of length $O(k)$ in σ'. (Longer backward arcs from comparisons with v that contradict these relations are now recognized as false and can be reversed.) The backward arcs incident to v create a new component that may include some components from σ and contains only $O(k^2)$ vertices.

We have now learned the complete comparison graph of $U \cup \{v\}$ by doing only $O(k \log n)$ comparisons. By Lemma 4 it remains to apply MB to σ', and to output the resulting order τ. Actually it suffices to optimize the component including v, since all other components were already optimal in σ and have not changed. At this point, $D(\tau)$ has at most k backward arcs (otherwise more than k faults exist, and we can stop), and their number is minimized. This establishes correctness of the insertion procedure.

In addition we need $F(k^2, k)$ time to optimize the new component of length $O(k^2)$, and $O(n)$ time to update the indices, due to the insertion of v. □

Lemma 7 is complemented with a simpler insertion procedure that we will apply first in our main algorithm (Theorem 4 below). The idea is to defer the appearance of backward arcs as long as possible, and until then the insertion steps are faster.

Lemma 8 *Given an order σ of U such that $D(\sigma)$ has no backwards arcs, we can construct an order τ such that $D(\tau)$ has no backwards arcs and: (i) either τ is an order of $U \cup \{v\}$, or (ii) τ is an order of $U \setminus \{u, u'\}$ for some $u, u' \in U$ where some comparison among u, u', v is false: Moreover, τ is obtained by using $O(\log n + k)$ comparisons and $O(n)$ time.*

Proof First we do usual binary search and temporarily believe the results. We insert v at the resulting position in σ. Note that all arcs of length 1 are forward. Only now we compare v to all vertices at distance at most $2k + 1$. If we get only forward arcs, then we define τ as the so obtained order of $U \cup \{v\}$. Otherwise we take some shortest backward arc. Since its length is not 1, it forms a directed triangle with two forward arcs. We remove the three involved vertices u, u', v and let τ be the remaining order. Trivially, some of the arcs in the directed triangle is false.

The number of $O(\log n + k)$ comparisons is obvious. We need $O(n)$ time to update the indices, and the time for all other operations is no larger. □

Theorem 4 *For $k < \sqrt{n}$, sorting with at most k recurring comparison faults can be accomplished with $O(n \log n + kn + k^2 \log n)$ comparisons.*

Proof We do fault-tolerant Insertion Sort, that is, beginning with the empty order we insert all n elements one by one in a minimum backward order. If k is small compared to n, actually the special case of no backward arcs is the more frequent one. In detail:

Phase 1: We apply Lemma 8 as long as possible. Since in total at most k comparisons are false and the removed triples of vertices (named v, u, u' in Lemma 8) are mutually disjoint, at most $3k$ vertices are removed from the order. We put these vertices aside. This needs $O(n \log n + kn)$ comparisons and $O(n^2)$ time.

Phase 2: We switch to Lemma 7 and insert the remaining $O(k)$ vertices. This needs $O(k^2 \log n)$ comparisons and $O(k^2 \log n + kn + kF(k^2, k))$ time. □

While the number of comparisons is already $O(n \log n)$ for any fixed k, this method would need $O(n^2 + k^2 \log n + kF(k^2, k))$ computations. As a final step we

will do the insertion procedures in a more economic way, to get rid of the $O(n^2)$ term.

7 Fault-Tolerant Treesort and Quicksort

For ease of presentation we did not pay much attention to the data structures so far. The downside of using an array for the order is that $O(n^2)$ time is needed, just for updating the indices n times. Of course, fault-tolerant Insertion Sort cannot be faster than in the error-free case. But we can also maintain the order and at the same time use a balanced search tree for the comparisons, as in Treesort. This does not affect the comparison graphs $D(\sigma)$ and accelerates the updates.

Theorem 5 *Sorting with at most k recurring comparison faults can be accomplished with $O(n \log n + kn + k^2 \log n)$ comparisons and with auxiliary computations requiring $O(n \log n + kn + k^2 \log n + kF(k^2, k))$ time.*

Proof We explain the modifications of the method from Theorem 4.

We maintain a partitioning of σ into buckets, which are sets of at least $2k + 2$ but at most $4k + 3$ consecutive vertices. The leftmost vertex of each bucket is the leading vertex. Since the leading vertices have distances larger than $2k + 1$, by Theorem 3, they appear in the correct order.

The leading vertices are stored in a balanced binary search tree. Instead of using indices for the positions of vertices in σ we use the search tree to find the appropriate position for insertion of the new vertex v. During Phase 1, in every node of the search tree we compare v to the leading vertex only. During Phase 2, in every node of the search tree we compare v to the leading vertex and its entire bucket. Since the buckets are larger than $2k + 1$, majority vote sends v in the correct direction (as we have seen before), and since the buckets have size $O(k)$, also the last comparisons during this search cost only $O(k)$ time. For every vertex we used $O(\log n + k)$ comparisons and $O(\log n + k)$ time in Phase 1, and $O(k \log n)$ comparisons and $O(k \log n + F(k^2, k))$ time in Phase 2.

Optimizing and re-ordering the $O(k)$-sized component of v affects only $O(1)$ buckets. If the new vertex v exceeds the size limit of buckets, we also split one bucket in two smaller ones. Then we update the search tree in $O(\log n)$ time. Altogether we get the claimed complexity bounds. □

A practical drawback of Treesort, already in the error-free case, is the overhead for tree manipulations which deteriorates the constant factor in the the time bound. Therefore we also present an alternative: to equip Quicksort with fault tolerance. Interestingly enough, it is possible to invoke our insertion procedure from Lemma 7 also in Quicksort. The reason why it works is that Quicksort divides an instance recursively in two smaller instances which, informally speaking, are independent in the error-free setting and interact only a little in the case of a few faults. In the following theorem, the (expected) complexity in O-notation is the same as for the deterministic algorithm, however the gain is in the hidden constant factor (similarly as for plain

Quicksort compared to other, deterministic sorting algorithms). We also remark that the expected $O(n \log n)$ bound for Quicksort holds for every instance, and the only randomness is in the choice of pivots; loosely speaking, there is no interference with the deterministic comparison faults.

Theorem 6 *Sorting with at most k recurring comparison faults can be accomplished with $O(n \log n + kn + k^2 \log n)$ expected comparisons and with auxiliary computations requiring $O(n \log n + kn + k^2 \log n + kF(k^2, k))$ expected time.*

Proof First remember how Quicksort works. A random pivot element p is compared to all other elements. A set L (R) collects all elements smaller (larger) than p, then L and R are sorted recursively, and L, p, R is the sorted order. In expectation this costs $O(n \log n)$ comparisons and time. Now, some extra work is needed due to possible comparison faults.

Instead of sorting L and R completely, we only produce minimum backward orders recursively. Since some comparisons with p may be false, some vertices in L should actually be in R and vice versa. We call them the dislocated vertices. Due to Theorem 3, dislocated vertices can only exist in a segment of length $O(k)$ at the right end of L and at the left end of R. Each of the $O(k)$ candidates v for a dislocated vertex in L is compared to the first $2k + 1$ vertices in R. If the majority claims that v is smaller, then Lemma 5 yields that v is actually smaller than all vertices of R, with $O(k)$ exceptions at the left end. In the other case we insert v in R as in Lemma 7. We proceed similarly with dislocated vertices in R. To turn the concatenation L, p, R into a minimum backward order, it remains to optimize the component of p.

Only $O(n/k)$ pivots are considered, because segments of length $O(k)$ are not further split recursively. The dislocation tests require $O(k^2)$ comparisons for every pivot, in total $O(kn)$. Since at most k vertices are dislocated in total (not only per pivot), all insertions together are done within a time bound $O(k^2 \log n + kF(k^2, k) + kn)$. For every pivot p, the component of p has length $O(k^2)$, thus in can be optimized in $F(k^2, k)$ time. We need to call an FPT algorithm at most k times, since every nontrivial component exists due to a comparison fault. Altogether the claimed expected complexity follows. □

8 Further Work: Decision Diagrams

We presented the first efficient algorithms for sorting with recurring faults. The methods are elementary but their combinations are not so obvious. For instance, it is unclear whether the approach of error detection and correction by majority voting would also work together with Mergesort. (It does work in [8], but in a different error model.) Simplicity should make the proposed algorithms practical at least for rare faults, i.e., when only small k are expected. For growing k however, the dependency on k, which is subexponential but still has k in the exponent, becomes an issue. Further research may find improved bounds, e.g., by more sophisticated Quicksort versions. Our aim in the present study was mainly to characterize the structure of the

solution space. There might also exist relationships to both permutation graphs and poset dimension, but we have not explored these directions.

Another concern is how the solution space can be represented. As we have seen in Section 3, there can exist up to 3^k MFAS which, up to further transpositions, describe all compatible orders. Their naive explicit enumeration can be avoided by decision diagrams, a well known kind of data structures that can compactly represent exponentially many objects in a natural way, as the system of paths in an layered (thus acyclic) directed graph. (See, e.g., [12] for an introduction.) To avoid confusion, we refer to vertices and arcs of decision diagram as *nodes* and *links*. A family of permutations of a set V (in our case, the compatible orders) can be represented by labeling the links by the elements of V in such a way that the sequence of labels along every directed path forms a permutation from that family.

We sketch a possible approach to construct decision diagrams for our problem. We may start from one compatible order $(v_1, \ldots, v_n)$, as obtained from fault-tolerant sorting. Suppose that we have already a decision diagram for the compatible orders of $\{v_1, \ldots, v_{j-1}\}$. (For $j = 1$ it consists of only one node.) We decide on the placement of v_j in the order of $\{v_1, \ldots, v_j\}$; note that it must be one of the last $2k+1$ positions. We update the last $2k + 1$ layers by inserting appropriate nodes and links with label v_j. Nodes can be merged if the incoming paths carry the same suffix of $2k+1$ labels, and the number of backward arcs within $\{v_1, \ldots, v_j\}$ is the same. (These numbers are stored at the nodes.) Therefore the width of the decision diagram depends only on k but not on n, whereas its length is n.

For example, in an order with non-overlapping backward arcs (see the end of Section 4), the decisions on the placement of elements are independent for elements from segments spanned by different backward arcs. Hence the nodes of the decision diagram need to memoize only the number of backward arcs decided so far, such that its width is k although the number of MFAS is exponential. For the general case with arbitrary backward arcs it would be interesting to get an upper bound on the width, smaller than the trivial bound 3^k. To this end one has to study branching rules for the reversal of arcs, and their characteristic polynomials (as in the analysis of search tree algorithms for FPT problems).

Let us call a pair of elements u and v *uncertain* if there exist two compatible orders where u is to the left of v, and v is to the left of u, respectively. We have seen that, in any compatible order, only pairs with distance at most $2k + 1$ are uncertain. Hence only $O(k^2 n)$ uncertain pairs exist. The decision diagram may be used to identify all uncertain pairs. An interesting question is whether there is a faster way.

Going one step further, using the decision diagram one may count, for any given u and v, the compatible solutions where u is to the left (right) of v. (Counting paths in an acyclic directed graph is straightforward.) In this way one could identify "informative" pairs where $u < v$ and $v < u$ holds in roughly half of the compatible orders. Now imagine a model similar to [10], where every pair allows a cheap and an expensive comparison. The cheap ones may have up to k faults, and the latter ones are always correct. In this scenario one may first do a cheap fault-tolerant sorting and then perform further expensive comparisons on informative pairs.

Regarding parameterized complexity, besides the total number k of faults one could introduce another parameter for the number of overlapping faults in the correct

order, that is, for elements $p < q < r < s$ such that comparisons would claim $r < p$ and $s < q$. This is motivated by the observed simplicity of the non-overlapping case, and by the natural assumption that comparison faults are more likely for elements being close in the correct order.

Acknowledgments The author would like to thank Stefan Funke for hinting to a related paper [10], the participants of IWOCA 2016 for further interesting suggestions, and the anonymous reviewers for a number of helpful comments.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Bessy, S., Fomin, F.V., Gaspers, S., Paul, C., Perez, A., Saurabh, S., Thomassé, S.: Kernels for feedback arc set in tournaments. J. Comput. Syst. Sci. **77**, 1071–1078 (2011)
2. Bodlaender, H.L., Boros, E., Heggernes, P., Kratsch, D.: Open problems of the Lorentz Workshop "Enumeration Algorithms using Structure". Techn. Report UU-CS-2015-016 Utrecht Univ (2015)
3. Braverman, M., Mossel, E.: Noisy Sorting without Resampling. In: Teng, S.H. (ed.) 19Th Annual ACM-SIAM Symp. Discrete Algor. SODA 2008, pp. 268–276 (2008)
4. Braverman, M., Mossel, E.: Sorting from noisy information. CoRR arXiv:0910.1191 (2009)
5. Cicalese, F.: Fault-Tolerant Search algorithms – reliable computation with unreliable information. Springer (2013)
6. Feige, U.: Faster FAST (Feedback arc set in tournaments). CoRR arXiv:0911.5094 (2009)
7. Felsner, S., Kant, R., Pandu Rangan, C., Wagner, D.: On the complexity of partial order properties. Order **17**, 179–193 (2000)
8. Finocchi, I., Grandoni, F., Italiano, G.F.: Optimal Resilient Sorting and Searching in the Presence of Memory Faults. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) 33Rd Int. Colloq. Autom. Lang. Program. ICALP 2006. LNCS, vol. 4051, pp. 286–298. Springer, Heidelberg (2006)
9. Fomin, F.V., Pilipczuk, M.: Subexponential Parameterized Algorithm for Computing the Cutwidth of a Semi-Complete Digraph. In: Bodlaender, H.L., Italiano, G. (eds.) 21St Annual Eur. Symp. Algor. ESA 2013. LNCS, vol. 8125, pp. 505–516. Springer, Heidelberg (2013)
10. Funke, S., Mehlhorn, K., Näher, S.: Structural filtering: a paradigm for efficient and exact geometric programs. Comput. Geom. **31**, 179–194 (2005)
11. Geissmann, B., Mihalák, M., Widmayer, P.: Recurring Comparison Faults: Sorting and Finding the Minimum. In: Kosowski, A., Walukiewicz, I. (eds.) 20Th Int. Symp. Fundam. Comput. Theory FCT 2015. LNCS, vol. 9210, pp. 227–239. Springer, Heidelberg (2015)
12. Knuth, D.E.: The Art of Computer Programming, Vol. 4A, Combinatorial Algorithms Part I. Addison–Wesley, Reading (2011)
13. Raman, V., Saurabh, S.: Parameterized algorithms for feedback set problems and their duals in tournaments. Theor. Comput. Sci. **351**, 446–458 (2006)
14. Rose, D., Lueker, G., Tarjan, R.E.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. **5**, 266–283 (1976)