The Solution Space of Sorting with Recurring Comparison Faults

Peter Damaschke

Department of Computer Science and Engineering
Chalmers University, 41296 Göteborg, Sweden
ptr@chalmers.se

Abstract. Suppose that n elements shall be sorted by comparisons, but an unknown subset of at most k pairs systematically returns false comparison results. Using a known connection with 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. An optimal parameterized enumeration algorithm for FAST also works for the more general chordal graphs, and this fact contributes to the efficiency of our representation. Then, 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.

1 Introduction

In the model of recurring faults in computations as introduced in [10], operations on certain items yield false results even when repeated. As opposed to transient or probabilistic failures, this model accounts for systematic errors. One of the problems investigated in [10] is to sort a set of n elements by comparisons, where at most k pairs return false comparison results; let us denote this assumption A_k. Recurring comparison faults can result from software bugs. One can also think of applications where the elements are real entities rather than data items in computer memory. For instance, archaeological 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 A_k from the comparison results only, since false but consistent answers might pretend any order. 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.

In [10], quality measures for alleged sorted sequences are defined and related to each other. This is done from the approximation point of view, asking: How much does an order obtained by doing *all comparisons* and some postprocessing differ from the unknown sorted order? What is not considered is the full solution space obtained from the comparisons, and the number of comparisons actually needed. A fault-tolerant search for the minimum element is provided, which returns an element of rank $O(k)$ by using $O(\sqrt{k}n)$ 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 this entire information? Specifically, how can we infer all comparison results by doing only a minority of them, ideally in a time close to $O(n \log n)$?

Starting from a version of the reversal lemma for minimal feedback arc sets (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 [11] for finding smallest MFAS in tournaments, called FAST. While a single minimum FAST can be computed faster, base 3 is optimal for explicit enumerations. Next, the MFAS enumeration characterizes the solution space, i.e., the orders compatible with all comparisons. While there can be $n^{O(k)}$ such orders, it suffices to know at most 3^k MFAS, as all other compatible orders are obtained from them by simple transposition sequences. Next we observe: If we know already a compatible order, we can certify it with only $O(kn)$ comparisons that form a chordal graph, hence the MFAS that describe all compatible orders can be enumerated in $O(3^k k^2 n)$ time. Finally we give efficient algorithms that actually find a compatible order and the information needed to reconstruct the solution space. A 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, that essentially need $O(n \log n)$ comparisons for 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.

Other related literature. As much work exists on fault-tolerant searching and sorting (see the survey [5]), it is important to pay attention to similarities. Liar models are also deterministic fault models with a maximum number of false answers, but they count repeated false answers, and the searcher can reconstruct the true results. Sorting in a model where some elements can be corrupted (but comparisons are correct) is considered in [8], where 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]. Enumeration problems find attention in various fields (see, e.g. [2]). The number of comparisons needed to decide properties of partial orders is studied in [7].

2 Preliminaries

Orders are ascending from left to right. We use the terms *vertex* and *element* interchangeably. Suppose that k is fixed. In a directed *comparison graph* $D = (V, A)$, where V is the set of the n elements to be sorted, every arc (u, v) indicates a comparison that claimed $u < v$. We call the arc (u, v) *true* if actually $u < v$, and *false* if $v < u$. 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. 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 [10] we call such 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. Every maximal sequence of this kind has length $d(\pi, \sigma)$ and ends in σ.

As usual, n and m denotes the number of vertices and arcs of a graph. 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 *topological order*, and one can construct a topological order or output a directed cycle in $O(n + m)$ time. For general $D = (V, A)$ we call an order σ of V a *minimal backward* order if no other order τ has $B(\tau) \subset B(\sigma)$. Hence, in acyclic graphs, minimal backward and topological orders are the same. A *minimum backward* order also has a minimum number of backward arcs.

A computational problem is fixed-parameter tractable (FPT) if instances of size n and with an additional input parameter k can be solved in $f(k) \cdot n^{O(1)}$ time, with some computable function f. A *feedback arc set (FAS)* 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 can use $F(n, k) = 2^{O(\sqrt{k})} n^{O(1)}$ [6, 9]. We will give time bounds in terms of $F(n, k)$, not in order to hide the exponential part, but in order to state the bounds in a generic way, independently of the current state of FAST.

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. A PEO is constructed in $O(n+m)$ time [12].

3 Characterizing and Enumerating MFAS

The "reversal lemma" was used in [11] and already discovered several times in the 1960s. It states that reversing the arcs of an MFAS makes a directed graph acyclic. The following extended version also considers 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. □

Theorem 1. *In a directed graph with chordal underlying graph, at most 3^k MFAS of at most k arcs exist, and they are 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. □

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 describe the family of all orders of a set being compatible with a comparison graph, by virtue of an MFAS enumeration and 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 k:

Enumerate all MFAS with 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, which is a compatible, minimal backward order (by Lemma 1). 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, but are arbitrary, subject to this condition. Equivalently, these orders have Kemeny distance at most $k - b$ from the minimal backward orders. We comment 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.

5 A Certificate for Sorting with Recurring Faults

The next natural question is whether we can get the solution space without doing all $O(n^2)$ comparisons when k is small, in view of the fact that sorting without faults (the case $k = 0$) needs only $O(n \log n)$ comparisons. Intuitively, we could first apply any $O(n \log n)$-time sort and then check the result and fix errors. However, usual sorting algorithms would not notice comparison faults, as they do not cause inconsistencies. They just continue and output a possibly false order σ. Another observation is that for any two neighbored elements u and v in σ we must actually do the comparison between u and v, since otherwise one could not tell whether $u < v$ or $v < u$. In order to spot errors we insert some redundancy, namely all arcs of length at most $2k+1$ in σ, and we do these $O(kn)$ comparisons. We do not take advantage of longer arcs from other comparisons possibly made before.

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

Suppose that $D(\sigma)$ has at most k backward arcs. In this case we are in a good position, as the next theorem says that further comparisons would not add more information, thus the instance of the sorting problem is then solved after $O(n \log n + kn)$ comparisons.

Theorem 3. *Consider an ordered comparison graph that contains all arcs of length at most $2k+1$, and 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 $u < v$.*

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) $u < w$ 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) $w < v$ 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 $u < w$ and $w < v$, 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 $u < w < v$. 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 other arcs without 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, in general we cannot expect to be lucky and get $D(\sigma)$ 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 point 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 point. A* trivial component *has only one vertex, and a* nontrivial component *has more than one vertex.*

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 $b := \sum_i b_i$. We define the following routine:

Procedure MB. In every nontrivial component C_i we compute a minimum FAS. Due to Lemma 1, topological sorting then yields a minimal backward order of C_i. We rearrange the vertices in every C_i accordingly.

Lemma 4. *The order 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, the number b is evident. 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

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. Now we want to insert another vertex $v \notin U$ and 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 elements ℓ and r with distance $O(k)$ in σ and $\ell < v < r$, 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 $O(k)$. □

Next we finalize the procedure. Recall the FAST time bound $F(n, k)$ from Section 2, the notion of components in 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. 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 with v create a new component that may incorporate some components from σ and contains only $O(k^2)$ vertices.

We have now learned the complete comparison graph of $U \cup \{v\}$ by actually doing only $O(k \log n)$ comparisons. By Lemma 4 it remains to apply MB to σ', and to take 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), these are the only true backward arcs in τ, and their number is minimized. This establishes correctness.

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 and faster procedure for the special case when backwards arcs did not yet appear. Then we can either insert another vertex as above, or recognize a fault.

Lemma 8. *Given an order σ of U such that $D(\sigma)$ has no backwards arcs, we can construct an order τ with the same property, such that either τ is an order of $U \cup \{v\}$, or τ is an order of $U \setminus \{u, u'\}$ for some $u, u' \in U$ where some comparison among u, u', v is false, 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 τ is the 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 and let τ be the resulting 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 are 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 pleasant, 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 catch with the use of an array for the order is that $O(n^2)$ time is needed only for updating the indices n times. Of course, fault-tolerant Insertion Sort cannot be faster than the error-free case. But we can also maintain the order and at the same time use a balanced search tree for the comparisons. 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 in $O(n \log n + kn + k^2 \log n + kF(k^2, k))$ 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 are in the correct order.

We store the leading vertices 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 drawback of Treesort 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 there. The reason why it works is that Quicksort divides an instance recursively in two smaller instances that are independent in the error-free setting and interact only a little in the case of a few faults. We formally state the theorem as follows, although the (expected) complexity in O-notation is the same as for the deterministic algorithm. We 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 our 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 in $O(n \log n + kn + k^2 \log n + kF(k^2, k))$ 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 in $O(k^2 \log n + kF(k^2, k) + kn)$ time. 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 asserted expected complexity follows. □

8 Conclusions and Further Work

We presented the first efficient algorithms for sorting with recurring faults. The methods are elementary but not obvious. It is unclear whether the approach of error detection and correction by majority voting would also work with Mergesort. (It works in [8], but in a different error model.) Simplicity should make it possible to implement the proposed algorithms, which was outside the scope of this study. We assumed small k, that is, applications with exceptional faults. For growing k, 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 was mainly to explore the structure of the solution space. In practice this does not necessarily mean that one must explicitly enumerate all compatible orders. Faults may appear independently in different segments, and then succinct enumerations of MFAS using binary decision diagrams can be much smaller. Some experimentation is needed to study the practicality.

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: SODA 2008, 268–276 (2008)
4. Braverman, M., Mossel, E.: Sorting from Noisy Information. CoRR abs/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 abs/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.) 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.F. (Eds.) ESA 2013. LNCS, vol. 8125, pp. 505–516, Springer, Heidelberg (2013)
10. Geissmann, B., Mihalák, M., Widmayer, P.: Recurring Comparison Faults: Sorting and Finding the Minimum. In: Kosowski, A., Walukiewicz, I. (Eds.) FCT 2015. LNCS, vol. 9210, pp. 227–239, Springer, Heidelberg (2015)
11. Raman, V., Saurabh, S.: Parameterized Algorithms for Feedback Set Problems and their Duals in Tournaments. Theor. Comp. Sci. 351, 446–458 (2006)
12. Rose, D., Lueker, G., Tarjan, R.E.: Algorithmic Aspects of Vertex Elimination on Graphs. SIAM J. Comp. 5, 266–283 (1976)