PROVABLE SPEEDUPS FROM LINKAGE LEARNING IN EVOLUTIONARY GRAPH OPTIMISATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph ordering problems, such as the NP-hard Feedback Arc Set (FAS) problem, are commonly attacked with evolutionary algorithms and greedy reinsertion heuristics. These methods search in permutation spaces and are guided solely by the objective value. As a result, they expose the immediate precedence constraints of the graph but leave transitive structure hidden. We show that this constitutes a genuine obstruction to efficient search. To this end, we show that even for a simple special case of FAS, namely to compute a topological ordering of a graph, an evolutionary algorithm (RLS) needs weakly exponential time. Building on linkage learning mechanisms developed for binary spaces, we transfer this machinery to permutation spaces and prove that it learns arcs of the underlying graph and never reports false ones. We then design a mutation operator that employs the learned arcs to identify sets of elements that are jumped together, thereby handling transitive structure implicitly. Equipping RLS with this operator, we show that learning and search reinforce each other: in every non-optimal step, the algorithm improves the fitness or enables the detection of a new arc with probability at least $1/n^2$. Hence, the improved algorithm orders any DAG in expected polynomial time without having to recover the entire graph. This exponential speedup demonstrates the usefulness of the enhanced operator in settings with hidden transitive structure. Experiments on instances with progressively removed transitivity information confirm this gap and indicate that the same operator remains effective on cyclic FAS instances.

1 INTRODUCTION

Evolutionary algorithms (EAs) are general-purpose optimisation methods that are frequently applied to *graph ordering problems* (Lugo et al., 2022; Newton et al., 2003; Segura et al., 2024), where the goal is to find a permutation of the vertices that optimises an objective function defined by the edges of a graph. A prominent example is the FEEDBACK ARC SET (FAS) problem, where the objective is to find an ordering that minimises the number of arcs pointing backwards. FAS is NP-hard and was among Karp's 21 initial NP-complete problems (Karp, 1972). Graph ordering problems also arise in applications such as scheduling and production planning.

Permutation-based EAs search directly in the space of vertex orderings and are typically guided by objective values. This raises a fundamental question: *what information about the graph can an evolutionary algorithm recover from objective values, and how does this information affect its ability to search efficiently?* A particularly important source of structure is *transitivity*. If a graph contains arcs $u \to v$ and $v \to w$, then every feasible ordering must place u before w, even if there is no arc from u to w. Thus, the graph explicitly specifies some precedence constraints while implicitly specifying many additional ones. Objective evaluations directly expose violations of explicit arcs, but the resulting transitive constraints may remain hidden from the optimiser.

We investigate whether this hidden structure can constitute a genuine obstruction to evolutionary search. As a clean test case, we consider *topological sorting*: given a directed acyclic graph (DAG), find a permutation in which every arc points from an earlier to a later vertex. Topological sorting is a fundamental graph problem and is the special case of FAS in which the optimum is zero. Although a topological ordering can be computed in linear time by standard graph algorithms, it provides a particularly well-defined setting for studying the information available to permutation-based EAs.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We first show that objective-based search can be inefficient even on this simple problem. Specifically, we prove that a well-known EA, *Randomised Local Search* (RLS), using jump mutations, requires weakly exponential expected time on a very simple DAG obtained by connecting two star graphs. The difficulty arises from the fact that the objective reveals the explicit precedence constraints but does not directly reveal their transitive consequences. This demonstrates that the representation of problem structure available to the optimiser can itself create a substantial search barrier.

Linkage learning aims to progressively identify dependencies between decision variables during the run of an EA. It enables the design of tailored variation operators that exploit dependencies between decision variables, thereby improving search performance (Dushatskiy et al., 2024; Przewozniczek & Komarnicki, 2020). Although linkage learning has been studied for a long time, it has recently found renewed interest (Dushatskiy et al., 2024; Goldman & Punch, 2014; Hsu & Yu, 2015; Przewozniczek & Komarnicki, 2020; Przewozniczek et al., 2021b). From a theoretical perspective, linkage learning has mainly been studied in binary spaces, while considerably less is known about its behaviour in permutation spaces, although there is a history of empirical work (see, e. g. Hsu & Yu (2015); Munetomo & Goldberg (1999); Thierens & Bosman (2011); Tinós et al. (2024)).

We adapt the *Non-Monotonicity Check* (NMC) (Munetomo & Goldberg, 1999; Przewozniczek et al., 2023) to permutations and show that it learns arcs of the graph without false positives. A new mutation operator uses them to select which elements are jumped together. As learning and search reinforce each other, RLS with this operator orders any DAG in expected polynomial time without recovering the entire graph, an exponential speedup over vanilla RLS.

Finally, experiments on instances with progressively removed transitivity information confirm this gap and indicate that our proposed mutation operator remains effective on cyclic FAS instances. Overall, our results show how hidden problem structure can obstruct evolutionary search in permutation spaces, and how learning arcs from objective values during search overcomes this barrier.

Related Work. Tailoring variation operators using problem knowledge, a vibrant research area in evolutionary computation, is termed *grey-box optimisation* (Whitley, 2019a;b). Przewozniczek et al. (2021a) introduced a linkage learning mechanism for permutations; it led to improved performance for GOMEA (a well-known EA) and a variant of the *Parameterless Population Pyramid* (P3) for permutations. Their mechanism is discussed in Section 4 and Appendix C. Goldman & Sudholt (2016) as well as Schmidbauer & Sudholt (2025) analysed the runtime of P3 on benchmark functions: concatenated traps and the hierarchical H-IFF problem, using different linkage-learning techniques. Schmidbauer & Sudholt (2025) showed that linkage learning builds truthful models that extract the maximum possible amount of linkage information from solutions seen so far. Qiao & Gallagher (2024) analysed a variant of GOMEA in a setting where linkages are known in advance. Studying the runtime of EAs in permutation spaces has recently received attention, sparked by the introduction of new test functions in Doerr et al. (2024). Opris et al. (2025) demonstrated that using crossover for permutations can yield an exponential speedup on a constructed problem. Baumann et al. (2024; 2025) considered a vertex ordering problem for minimising the number of edge crossings in graph drawings. It generalises the classical sorting problem, and on some instance classes it corresponds to computing a topological ordering on the input graph. Finally, Branson & Sutton (2023) designed and analysed grey-box EAs for the related k-FEEDBACK VERTEX SET problem.

2 PRELIMINARIES

Let $[n] = \{1, \dots, n\}$ and let V be a set of n elements. A *permutation* of V is a bijective map $\pi\colon [n] \to V$. We write π_i for $\pi(i)$ and π_u^{-1} for $\pi^{-1}(u)$. A permutation π defines an ordering of V via $u <_\pi v$ if and only if $\pi_u^{-1} < \pi_v^{-1}$ for $u, v \in V$. For a set $S \subseteq V$ we write $S <_\pi v$ ($v <_\pi S$) if $s <_\pi v$ ($v <_\pi s$) for all $s \in S$. We denote a permutation π as a sequence of its elements $\pi = \langle \pi_1, \pi_2, \dots, \pi_n \rangle$, e.g., $\pi(1) = 2, \pi(2) = 3, \pi(3) = 1$ is written as $\pi = \langle 2, 3, 1 \rangle$. Two elements u, v in π are *consecutive* if $|\pi_u^{-1} - \pi_v^{-1}| = 1$. The set of all permutations of V is denoted $\Pi(V)$.

A *directed acyclic graph (DAG)* is a directed graph that does not contain a cycle. For a directed graph $G = (V, E)$ a set $F \subseteq E$ is called a *feedback arc set* if $G - F$ is a DAG. We call an arc $e = (u, v) \in E$ *bidirected* if also $(v, u) \in E$. A directed graph is *weakly connected* if its underlying undirected graph is connected. Given a directed graph, the FEEDBACK ARC SET problem (FAS) asks for a feedback arc set of minimum size. Let $G = (V, E)$ be a directed graph, and let π be a



Figure 1: Double-star instance (left) and a locally optimal ordering of the vertices (right).

permutation of V. An arc $(u, v) \in E$ is a *forward arc* (with respect to π) if $u <_\pi v$ and a *back arc* if $v <_\pi u$. A permutation of V without back arcs is a *topological ordering*. It is well known that a graph G is a DAG if and only if it admits a topological ordering. In particular, if F is a minimal (w.r.t. inclusion) feedback arc set, then a topological ordering of $G - F$ determines a permutation of V whose set of back arcs is precisely F. Thus FAS is equivalent to determining a permutation of V that minimises the number of back arcs. We therefore define the objective function $f(\pi)$, to be minimised, as the number of back arcs with respect to π. Formally, letting $A = (a_{u,v})_{u,v \in V}$ denote the adjacency matrix of G,

$$f(\pi) \coloneqq \sum_{u <_\pi v} a_{v,u}. \tag{1}$$

This is a special case of the *linear ordering problem*, which asks for a permutation that minimises the same objective, but $A \in \mathbb{R}^{n \times n}$ may be an arbitrary matrix. The optimal objective value is 0 iff G is a DAG. Thus, finding an optimal permutation corresponds to computing a topological ordering.

Following Baumann et al. (2024), to be able to draw meaningful conclusions, we reduce complexity in the algorithm design by focusing on a simple randomised local search that evolves a single candidate solution by creating a single offspring. Ties are broken in favour of offspring to enable the EA to search amongst solutions of equal fitness. The offspring is created using the *jump* operator, which theory and practice suggest is the most useful operator for order-based permutation problems (Baumann et al., 2024; 2025). Given $i, j \in [n]$ the jump operation repositions the vertex at position i to position j by shifting all elements between i (exclusive) and j (inclusive) towards i by one. If not stated otherwise, i and j are chosen uniformly at random with $i \neq j$. For convenience, we say that we jump two elements u, v, when we perform the respective operation to the positions π_u^{-1} and π_v^{-1}. A scheme for this algorithm is shown in Algorithm 1.

Algorithm 1: RLS for permutation-based optimisation.

```
Choose a permutation π uniformly at random
while stopping criterion not met do
    create π′ by copying π and performing a jump
    if f(π′) ≤ f(π) then π ← π′
```

We shall refer to the permutation in iteration t as the *current permutation* $\pi^{(t)}$, with $\pi^{(0)}$ being the initial permutation. In this context, $<_t \coloneqq <_{\pi^{(t)}}$ denotes the ordering of the elements according to the current permutation and "the current number of back arcs" also refers to the current permutation.

The optimisation time is the number of function evaluations until a global optimum is discovered. A function of n is *weakly exponential* if it grows at least as fast as $e^{\Omega(n^\varepsilon)}$ for a constant $\varepsilon > 0$.

3 Plain RLS fails on graphs without transitive edges

Baumann et al. (2025) showed that sifting—a greedy heuristic that always performs the best jump instead of a random one—can sort transitive graphs in polynomial time. As we show next, this is also true for the simpler RLS, albeit with linear overhead. The full proof can be found in Appendix A.

Theorem 3.1. *Let G be a DAG containing all transitive edges, with n vertices and m edges. Then, RLS optimises G in an expected number of $O(n^2 m)$ generations.*

In contrast, we show that simple graphs without transitive edges can be very hard for RLS. The main result of this section is that there is a DAG on which RLS requires weakly exponential time. This implies that the worst-case expected optimisation time on the class of DAGs is weakly exponential.

The *double-star* graph D_n, for n vertices, $n \geq 4$ even, is a DAG with two centre vertices ℓ, r and $(n-2)/2$ vertices in the sets L, R, each. Each vertex of L has one arc to ℓ, and r has $(n-2)/2$ arcs, one to each vertex of R. Additionally, there is an arc from ℓ to r, cf. Figure 1, left. All permutations in which $L < \ell < r < R$ are global optima. We call vertices in R *red* and vertices in L *blue*. The fitness function gives strong hints to place r before red vertices, as there are many arcs from r to vertices in R. Likewise, it gives strong hints to place ℓ after blue vertices in L. Intuitively, if we start with a uniformly random permutation, the fitness gain of ℓ moving far to the right and r moving far to the left is typically high, as long as L and R are mixed in the permutation: when $\ell < r$, jumping ℓ over r turns the arc (ℓ, r) into a back arc, but for each vertex in L that is jumped over, the number of back arcs reduces by one. In the beginning, it is thus very likely that ℓ is jumped very far to the right and r is jumped very far to the left, creating a local optimum in which (ℓ, r) is the only back arc. In this set of local optima, all vertices of L are placed left of ℓ and all vertices in R are placed right of r; see Figure 1 right. As there are no arcs between L and R, they can mix arbitrarily in the region between r and ℓ. In particular, the fitness function gives no hints to separate the vertices in R and L from one another. However, such a separation is necessary to reach a global optimum.

Theorem 3.2. *The expected optimisation time of RLS on the double-star D_n is at least $e^{\Omega(n^{1/3})}$.*

Proof Sketch (Full proof in Appendix B). With probability $\Omega(1/n^2)$, RLS starts with a permutation $\pi^{(0)}$ in which r is placed at position 1, and ℓ is placed at position n. We call this event A and, denoting the optimisation time by T, we will show $\mathrm{E}(T \mid A) \geq e^{\Omega(n^{1/3})}$. The sought expectation is

$$\mathrm{E}(T) = \mathrm{E}(T \mid A)\Pr(A) + \mathrm{E}(T \mid \overline{A})\Pr(\overline{A}) \geq e^{\Omega(n^{1/3})} \cdot \Omega(1/n^2) = e^{\Omega(n^{1/3})}$$

by the law of total probability. To bound $\mathrm{E}(T \mid A)$, we define the distance $\Delta^{(t)}$ between the positions of ℓ and r at time t. Initially, $\Delta^{(0)} = n - 1$, and every global optimum has a negative distance.

In steps where neither r nor ℓ is selected to be jumped, $\Delta^{(t)}$ may change by ± 1 as jumps may shift either r or ℓ. The distance increases if an element left of r or right of ℓ is jumped to a position between r and ℓ; all such jumps are accepted (cf. Lemma B.1). The distance decreases if an element between r and ℓ is jumped left of r or to the right of ℓ; however, such a jump is only accepted for vertices of one specific colour. This induces a bias towards increasing distance (cf. Lemma B.2).

A major complication arises from mutations in which r or ℓ itself is chosen to jump, as the offspring may have a drastically smaller distance than its parent. Fortunately, a jump of r to the right is only accepted if it does not skip any red vertex (or if it only skips one red vertex, in case r jumps beyond ℓ). In contrast, jumps of r to the left are always accepted. Whether a rightward jump of r is accepted, therefore, depends on the colours of the succeeding vertices. These colours can be mixed uniformly, in which case only small jumps are possible. However, if r is succeeded by a contiguous block of blue vertices, large jumps of r may occur. Such a situation can arise, for example, if r makes a significant jump to the left, skipping over many blue vertices, and is then selected for another jump to the right. Hence, we need to carefully analyse the dynamics of elements between r and ℓ.

Our proof exploits the following observation. Since all jumps to a position between r and ℓ are accepted, and red and blue vertices are equally likely to be selected for a jump, any fixed block of indices receives a constant influx of both red and blue vertices. As a result, blocks containing only vertices of one colour are quickly disrupted and replaced by a mixture of red and blue vertices. This mixing frequently occurs during periods in which r and ℓ are not being jumped (cf. Lemma B.3).

We show that the algorithm frequently returns to *safe* permutations in which r rests among the first ξn positions, ℓ among the last ξn positions, and both remain there long enough for the elements at positions in $[\xi n, n - \xi n]$ to become mixed, for some constant $\xi > 0$ (cf. Lemma B.4). Following Sudholt (2021), we refer to this as a *fallback*. To escape from a safe permutation, the algorithm must make progress via small jumps, as it cannot jump over blocks that contain both red and blue vertices (cf. Lemma B.5). During this time, no fallback may occur, as this would immediately eliminate all progress made so far. We show that the probability of avoiding a fallback until a global optimum is found is weakly exponentially small. The probability of this happening in weakly exponential time is still weakly exponentially small, which establishes $\mathrm{E}(T \mid A) = e^{\Omega(n^{1/3})}$ as claimed. □

Table 1: Properties for detecting linkage with NMC for bit strings (left) and permutations (right).

(B1)	$f(x) < f(x_i)$ and $f(x_j) \geq f(x_{ij})$	(C1)	$f(\pi) < f(\pi_{u\to v})$ and $f(\pi_{u\leftarrow v}) \geq f(\pi_{u\leftrightarrow v})$
(B2)	$f(x) = f(x_i)$ and $f(x_j) \neq f(x_{ij})$	(C2.1)	$f(\pi) = f(\pi_{u\to v})$ and $f(\pi_{u\leftarrow v}) < f(\pi_{u\leftrightarrow v})$
		(C2.2)	$f(\pi) = f(\pi_{u\to v})$ and $f(\pi_{u\leftarrow v}) > f(\pi_{u\leftrightarrow v})$
(B3)	$f(x) > f(x_i)$ and $f(x_j) \leq f(x_{ij})$	(C3)	$f(\pi) > f(\pi_{u\to v})$ and $f(\pi_{u\leftarrow v}) \leq f(\pi_{u\leftrightarrow v})$
(B4)	$f(x) < f(x_j)$ and $f(x_i) \geq f(x_{ij})$	(C4)	$f(\pi) < f(\pi_{u\leftarrow v})$ and $f(\pi_{u\to v}) \geq f(\pi_{u\leftrightarrow v})$
(B5)	$f(x) = f(x_j)$ and $f(x_i) \neq f(x_{ij})$	(C5.1)	$f(\pi) = f(\pi_{u\leftarrow v})$ and $f(\pi_{u\to v}) < f(\pi_{u\leftrightarrow v})$
		(C5.2)	$f(\pi) = f(\pi_{u\leftarrow v})$ and $f(\pi_{u\to v}) > f(\pi_{u\leftrightarrow v})$
(B6)	$f(x) > f(x_j)$ and $f(x_i) \leq f(x_{ij})$	(C6)	$f(\pi) > f(\pi_{u\leftarrow v})$ and $f(\pi_{u\to v}) \leq f(\pi_{u\leftrightarrow v})$

4 Enhancing RLS with Linkage Learning

In this section, we enhance RLS with a linkage learning concept. Inspired by linkage learning techniques for pseudo-Boolean optimisation, we introduce a linkage detection method for permutation-based problems that encodes linkage information in a *dependency structure matrix (DSM)*. Then, we propose a modified jump operator, called *DSM-jump*, which exploits the linkage information in a DSM. Finally, we show that the resulting algorithm DSM-RLS, which in each generation performs linkage detection and applies a random DSM-jump operation, computes a topological ordering of any DAG in expected polynomial time.

4.1 Linkage learning for ordering problems

Przewozniczek et al. (2021a) proposed a linkage definition for permutation-based problems. Unfortunately, this definition turns out to be rather restrictive in our setting, which we discuss in more detail in Appendix C. For this reason, we propose an alternative linkage detection mechanism inspired by the *Non-Monotonicity Check* (NMC) (Munetomo & Goldberg, 1999; Przewozniczek et al., 2023), a linkage detection mechanism for pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$. We first explain the NMC for pseudo-Boolean functions, which map bit strings to real numbers. The NMC attempts to detect linkage between two decision variables, referred to by their corresponding bits $i \neq j \in [n]$, by comparing $f(x), f(x_i), f(x_j)$, and $f(x_{ij})$, where x_i, x_j, x_{ij} denote the bit string obtained by flipping bit(s) i, j, or i and j in x, respectively. Two bits i and j are *x-linked* if at least one of the properties (B1)–(B6) holds, see Table 1 (left).

The intuition is that the sign of the objective value change induced by flipping bit i is different before and after flipping bit j (or vice versa). To remove the dependency on x, two bits $i, j \in [n]$ are defined to be *linked* if they are *x-linked* for some $x \in \{0,1\}^n$.

NMC for Ordering Problems. We adapt this definition to the setting of permutations of a set V with an objective function $f\colon \Pi(V) \to \mathbb{R}$ as follows. First, since we are studying ordering problems, our main concern is the order of pairs of elements rather than their exact positions. We hence aim to detect dependencies between pairs of elements from V. For a permutation $\pi \in \Pi(V)$ and elements $u, v \in V$, let $W_\pi(u,v)$ denote the elements between u and v, including u and v. We let $\pi_{u\to v}$, $\pi_{u\leftarrow v}$ and $\pi_{u\leftrightarrow v}$ be the permutation obtained from π by jumping u to the position of the element in $W_\pi(u,v)$ consecutive to v, jumping v to the position of the element in $W_\pi(u,v)$ consecutive to u, and exchanging u and v, respectively; see Figure 2. Note that performing both jumps across the block corresponds to the exchange of u and v, unless u and v are consecutive, when in fact $\pi_{u\to v} = \pi_{u\leftarrow v} = \pi$. Similarly to the pseudo-Boolean case, we detect dependencies between u and v by comparing the objective values of these four permutations. We define u and v to be *π-linked* if at least one of the properties (C1)–(C6) holds, see Table 1 (right).

We define u and v to be *linked* if there is a permutation π such that they are π-linked. We now give some intuition on how the conditions above define a preference. We do so only for the first two cases since the other cases work similarly. In case C1, jumping u next to v worsens the objective (recall that we seek to minimise f), but, assuming that v has already been jumped next to u, that same jump becomes beneficial. In both situations u jumps over all elements in $W_\pi(u,v) \setminus \{u,v\}$,

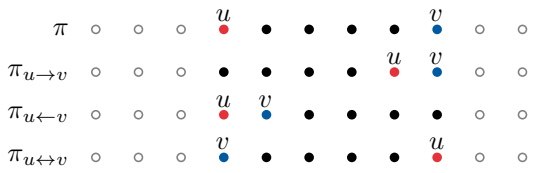


Figure 2: Illustration of the permutations used for detecting linkage between two elements u and v with respect to a permutation π. The elements in W are solid, elements outside of W are hollow.

but in the latter situation it additionally jumps over v, which must therefore be the cause of the difference. Thus, this case indicates that the relative order of u and v should be changed. In case C2.1, jumping u next to v is neutral, but, assuming that v has already been jumped next to u, the same jump worsens the objective. This indicates that u and v should preserve their relative order.

We define the corresponding DSM $D(\pi) = (d_{u,v}(\pi))_{u,v\in V}$ by setting $d_{u,v}(\pi) = 1$ if and only if, $u <_\pi v$ and one of C2.1, C3, C5.1, or C6 holds, or $v <_\pi u$ and one of C1, C2.2, C4, or C5.2 holds. Otherwise, $d_{u,v}(\pi) = 0$. Further, for a set of permutations $\Pi \subseteq \Pi(V)$, we let $D(\Pi)$ denote the component-wise maximum of the matrices $D(\pi)$ for $\pi \in \Pi$.

4.2 A DSM-BASED SEARCH ALGORITHM

We are now ready to implement linkage learning into RLS. To this end, we first propose a new mutation operator that uses the learned linkages. Afterwards we discuss how to integrate the linkage learning algorithm into the optimisation process.

DSM-based jump operator. We assume that we have access to a DSM $D = (d_{u,v})_{u,v\in V} \in \{0,1\}^{n\times n}$, where $d_{u,v} = 1$ indicates a preference that u should be before v. We are not aware of any existing mutation operators for permutations that exploit linkage information. Hence, we follow the typical design principles of linkage learning operators: based on a randomly chosen main operation, we identify relevant elements using the DSM and perform additional modifications on them, provided these do not worsen the objective value. Examples of this approach exist both for the pseudo-Boolean case (Tinós et al., 2022; 2024), where the bits perturbed in a generation are chosen from the neighbourhood defined by the DSM, as well as for crossover operators (Dushatskiy et al., 2024; Hsu & Yu, 2015; Przewozniczek et al., 2021a; Tinós et al., 2015).

Let π be a permutation of V. A *DSM-path* is a sequence $p = \langle w_1, \dots, w_k\rangle$ that is *aligned* in π and in D, that is, we have $w_i <_\pi w_{i+1}$ and $d_{w_i,w_{i+1}} = 1$ for all $i \in [k-1]$. We say that p is a DSM-path from w_1 to w_k. The idea is that, if an operation jumps w_1 to a position behind w_k (resp. w_k to a position before w_1), then p in a sense opposes this modification and we thus identify w_k (resp. w_1) as a relevant element for such an operation. We then attempt to apply the same jump to each of the relevant elements in such a way that their relative order is preserved.

More precisely, our mutation operator *DSM-jump* works as follows. When applied to a permutation π, it first picks two positions i and j uniformly at random. Let u and v denote the elements at positions i and j, respectively, and let $W^-_\pi(u,v)$ denote the elements strictly between u and v. The main operation is then jumping u to the position of v. This changes not just the relative order of u and v but also the relative order of u and w for each $w \in W^-_\pi(u,v)$. To remedy this, our DSM-jump operator first performs the main operation and then scans the elements it jumped over in the direction from v to u in π for relevant elements, which have a DSM-path that opposes the main operation and jumps each such element w' to the new position of u as well, provided this does not worsen the fitness. We give pseudocode for the procedure in Algorithm 2 and an example in Figure 3. We refer to the jump of u to v as the *main jump* and to the remaining jumps that are attempted based on the linkage information as *repair jumps*.

DSM-based RLS. The algorithm DSM-RLS extends RLS as follows. We initialise a DSM $D^{(0)} = (d^{(0)}_{u,v})_{u,v\in V} \in \{0,1\}^{n\times n}$ with $D(\pi^{(0)})$ and in each iteration attempt to learn linkages from the current permutation $\pi^{(t)}$, using NMC to obtain $D^{(t)}$ from $D^{(t-1)}$. Afterwards, we apply the DSM-jump operator using $D^{(t)}$ and accept the offspring if it is no worse; see Algorithm 3.

Algorithm 2: DSM-jump for minimising f

Input: permutation π, DSM D

$\sigma \leftarrow$ jump i, j chosen uniformly at random from $[n]$ in π /* main jump */
if $(j < i)$ **then** /* π_i jumps to the left */
  for $p = j$ ***to*** $i - 1$ **do**
    if *there is a DSM-path from* π_p *to* π_i *in* π **then**
      $\sigma' \leftarrow$ jump $p + 1, j$ in σ /* repair jump */
      if $(f(\sigma') \leq f(\sigma))$ **then** $\sigma \leftarrow \sigma'$ and $j \leftarrow j + 1$
else /* π_i jumps to the right (symmetric) */
  for $p = j$ ***downto*** $i + 1$ **do**
    if *there is a DSM-path from* π_i *to* π_p *in* π **then**
      $\sigma' \leftarrow$ jump $p - 1, j$ in σ /* repair jump */
      if $(f(\sigma') \leq f(\sigma))$ **then** $\sigma \leftarrow \sigma'$ and $j \leftarrow j - 1$
return σ

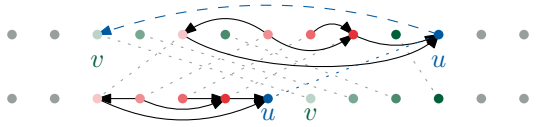


Figure 3: Illustration of a DSM-based jump of element u to the position of v before (top) the DSM-jump and after (bottom) the DSM-jump. Elements that have different positions in the top and the bottom are connected by dotted lines. Relevant elements for the DSM-jump are drawn in red ink. A black arrow (w, x) indicates a linkage between w and x: $d_{wx} = 1$.

4.3 Runtime analysis

In this section, we show that DSM-RLS can topologically order DAGs in expected polynomial time. The following lemma characterises the DSMs our linkage learning algorithm produces.

Lemma 4.1. *Let $G = (V, E)$ be an unweighted graph with adjacency matrix $A = (a_{u,v})_{u,v \in V}$ and let f be the objective function from Eq. (1). Let further π be a permutation of V, $u, v \in V$. We then have $d_{u,v}(\pi) = 1$ if and only if $a_{u,v} - a_{v,u} = 1$ and $\left[\sum_{w \in W_\pi^-(u,v)} (a_{w,u} - a_{u,w}) \in \{0, 1\}$ or $\sum_{w \in W_\pi^-(u,v)} (a_{w,v} - a_{v,w}) \in \{0, -1\} \right]$.*

The lemma states that for any DSM D learned using NMC, we have that $D \subseteq A$, where A is the adjacency matrix of G. Further, NMC only learns the linkage between u and v in π, if the absolute change in the objective value of jumping u or v over the block $W_\pi^-(u, v)$ is at most 1. In other words, if the block between u and v has a strong interaction with both vertices, this hides their linkage. On the other hand, for any vertex v linkage is surely discovered to the two closest vertices left and right of v in π that are neighbours of v in G, if they exist.

We now analyse the effect of DSM-jump operations. The following lemma shows that, if we DSM-jump an element u to an element v, and the used DSM D contains a sufficient amount of linkage information for the relevant elements between u and v, then this jump is accepted together with all repair jumps. The required amount of linkage information is quantified by comparing the linkage information contained in D to the ground truth given by the adjacency matrix of the input graph.

Algorithm 3: DSM-RLS

Choose a permutation π uniformly at random
$D \leftarrow D(\pi)$
while *stopping criterion not met* **do**
  $D \leftarrow \max\{D, D(\pi)\}$ /* component-wise maximum */
  $\pi' \leftarrow$ DSM-jump(π, D)
  if $f(\pi') \leq f(\pi)$ **then** $\pi \leftarrow \pi'$

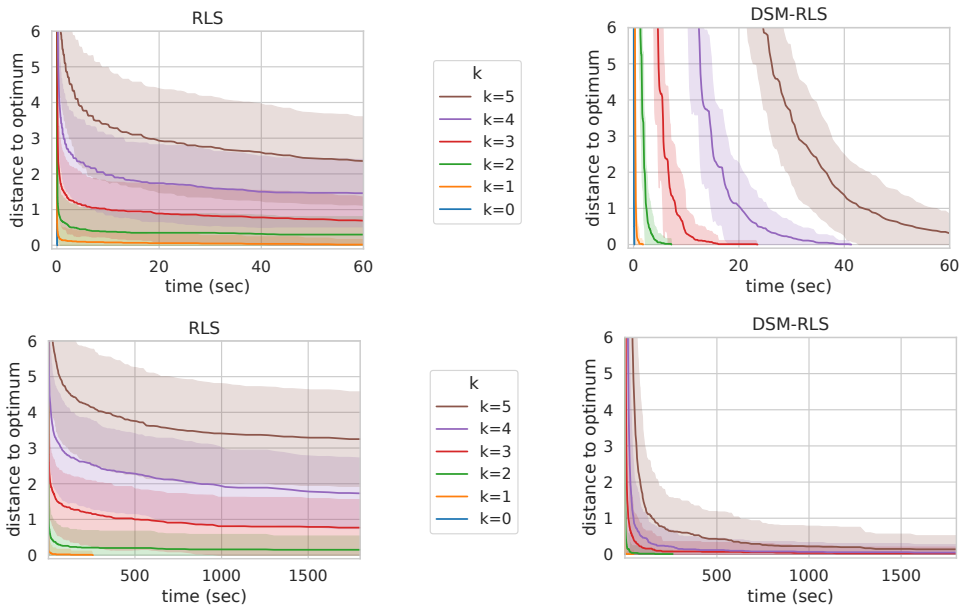


Figure 4: Top: random trees with 40 vertices before adding k subdivisions. Bottom: uniform graphs with 20 vertices before adding k subdivisions. The shaded areas show the standard deviation. Executions were stopped after the optimum was found or the time limit was reached.

Lemma 4.2. *Let $G = (V, E)$ be a DAG with adjacency matrix A and let $D \subseteq A$ be a DSM. Let $u, v \in V$ and $\pi \in \Pi(V)$. If all elements $x \in W_\pi^-(u, v) \cup \{v\}$ that have a DSM-path from or to u with respect to A and π also have such a DSM-path with respect to D and π, then the DSM-jump with main jump from u to v is accepted together with all repair jumps.*

We now prove that, if the underlying graph is a DAG, DSM-RLS solves the problem to optimality.

Theorem 4.3. *DSM-RLS computes a topological ordering of any DAG G with n vertices and m edges in expected $O(n^2 m)$ iterations and $O(n^4 m)$ fitness evaluations.*

Proof Sketch (Full proof in Appendix C). We show that if the algorithm has not found an optimal solution, it has a chance of at least $1/n^2$ to improve the fitness or to detect a new linkage. The main idea is to consider a distance-wise closest pair of vertices u, v, with a back arc in the current permutation $\pi^{(t)}$. If all linkage information between u and the elements of $W^-_{\pi^{(t)}}(u, v)$ is known, jumping u to v is accepted and improves the fitness by Lemma 4.2. Otherwise, we use Lemma 4.2 to argue that a DSM-jump, whose main jump moves u right next to the first vertex of $W^-_{\pi^{(t)}}(u, v)$ and whose linkage with u is not yet known, will be accepted and that the corresponding linkage is then found in the next iteration by Lemma 4.1. Finally, as there are at most m back arcs and missing linkage entries, respectively, we obtain an upper bound of $O(n^2 m)$. □

5 Experimental Evaluation

To complement our theoretical findings and to validate the effect of the proposed DSM-jump operator, we empirically compare RLS to DSM-RLS. As Theorem 3.1 and Theorem 3.2 show, the jump operator struggles when transitive information about the instance is hidden. To confirm this empirically, we evaluated their performance on k-subdivisions of two classes of graphs, *random trees* and *uniform graphs*. A *k-subdivision* of a graph replaces every edge by a directed path of length $k + 1$. We use subdivisions to increase the ratio of missing transitive arcs and therefore to "hide" structural information encoded in the fitness function. A random tree is generated by starting from a single vertex and iteratively adding a new vertex that is connected to or from one of the existing vertices uniformly at random. A uniform graph is obtained by starting with an edgeless graph on

a fixed number of vertices and adding for each vertex pair an arc independently with probability $1/4$ and directing it equally likely in either direction. Note that the latter graphs are not necessarily acyclic and yield FAS instances. We generated 100 random trees with initially 40 vertices and 100 uniform graphs with initially 20 vertices and computed k-subdivisions for $k \in \{0, \dots, 5\}$. We ran both algorithms on the random trees for $60\,\mathrm{s}$ and on the more difficult uniform graphs for $1800\,\mathrm{s}$. Experiments were run on a single core of an AMD EPYC 7443P processor with 2.85 GHz and 2 GB of RAM. Figure 4 shows the fitness development over time. The curves show the averages and the standard deviations for the 100 runs for each value of k.

For random trees, a single subdivision is manageable for RLS, but the distance to the optimum increases with k, and RLS stagnates. DSM-RLS solves the majority of the instances within the time limit for all values of k, although the running time increases significantly with k. A similar picture emerges for the uniform graphs, where the running times become even worse for larger k. While for $k = 0$ both algorithms quickly find the optimum, RLS stagnates for $k = 2$ with increasing gaps for higher values. In contrast, DSM-RLS typically finds the optimum within the time limit. We note that Appendix D contains more information on success rates and exact running times of the same experiments, confirming the above conclusions.

The primary drawback of DSM-RLS is its per-generation execution time, which is significantly higher than that of RLS. As more detailed analyses in the appendix reveal, this overhead is driven predominantly by the exhaustive application of the NMC to all vertex pairs—a design choice intended to simplify the theoretical analysis. Further, on these sparse instances, most of the DSM is already discovered during the first iteration. It thus seems safe to significantly reduce the number of DSM checks and thus to drastically decrease execution time of a single generation. This can be achieved for example by skipping DSM-checks in a large fraction of the generations or by performing linkage learning only on components modified during mutation (as common for evolutionary algorithms using local search (Tinós et al., 2024)).

Limitations. The experiments reveal that missing transitive structure forms a real obstruction for the optimisation process of RLS whenever transitive information is hidden by subdivided edges (or path-like structures). The proposed linkage learning EA can successfully handle such instances even if the base graph is cyclic, indicating a potential use-case for general (weighted) FAS. However, due to the limited size and complexity of the studied cyclic instances and the fact that both algorithms can solve the base case without subdivisions to optimality consistently, the usefulness in these settings remains largely unexplored. Furthermore, the proposed relaxation to linkage checks needs further investigation to ensure that necessary linkage is detected.

6 Conclusions

We investigated the runtime of evolutionary algorithms for graph ordering problems, focusing on the role of transitive information. Although jump-based and related reinsertion operators are widely used for order-based permutation problems (He & Sýkora, 2004; Lugo et al., 2022; Matuszewski et al., 1999; Simpson et al., 2016), we showed that they can be inefficient even for computing a topological ordering of a simple DAG. In particular, Theorem 3.2 shows that RLS using jumps requires weakly exponential expected time when transitive information is not explicitly available.

We then showed how learning arcs during the search can overcome this barrier. We transferred the Non-Monotonicity Check (NMC) to permutation spaces and proved that it learns arcs of the input graph without false positives. Based on these arcs, DSM-jumps reposition elements linked to the moved element, thereby handling transitive structure implicitly. As learning and search reinforce each other, DSM-RLS computes a topological ordering of any DAG in expected $O(n^2 m)$ generations without recovering the entire graph, yielding an exponential speedup over jump-based RLS.

Our results identify hidden transitive structure as a source of difficulty for evolutionary search on graph ordering problems and show that linkage learning can remove the resulting barrier. Experiments with progressively removed transitivity information support the theoretical results and indicate that DSM-RLS remains effective on cyclic FAS instances. The same ideas may extend to other linear ordering problems, including general Feedback Arc Set. Our experiments provide an initial indication of this potential, but are insufficient for general conclusions. Theoretical guarantees and broader empirical evaluation for FAS remain directions for future work.

AI USE STATEMENT

All proofs were conceived and first written by the authors. We then used generative AI tools to check the proofs and the paper for errors. This identified several errors (including an incorrect case distinction in Lemma 4.1 and an invalid probability bound in the proof of Lemma B.5), and the tool suggested corrections, which the authors verified and incorporated; all final proofs were checked by the authors. We further used generative AI tools to validate the reproducibility of our work (inspecting the source code in our repository and preparing usage instructions), to improve readability, to identify relevant literature, to format references, and to propose a title and keywords. We did not use generative AI to generate synthetic datasets, propose hypotheses, design methodology or experiments, implement methods or interpret results; translation, data cleaning and qualitative or thematic data analysis are not applicable to this work. We reviewed all AI-assisted work and take full responsibility for the final content, including all text, claims and artefacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The proofs to all mathematical claims and theorems can be found in the appendix. We provide our implementation as part of the submission together with a readme file explaining how the Java code should be used. We also provide the randomly generated instances. The pseudocode and all necessary information to reproduce the algorithms are given. The setup of the experiments is comparatively simple and the way the instances are created as well as how the plots are created is precisely described. To ensure that all necessary information can be found in the main body, we fed the relevant information of the paper to Claude Sonnet 4.6, which was able to reproduce the results on its own, i.e., to fully implement both algorithms in Python, correctly generate the instances and to reproduce similar plots.

REFERENCES

Jakob Baumann, Ignaz Rutter, and Dirk Sudholt. Evolutionary computation meets graph drawing: Runtime analysis for crossing minimisation on layered graph drawings. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '24)*, pp. 1533–1541. ACM, 2024.

Jakob Baumann, Ignaz Rutter, and Dirk Sudholt. Analysing the effectiveness of mutation operators for one-sided bipartite crossing minimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '25)*, pp. 872–880. ACM, 2025.

Luke Branson and Andrew M. Sutton. Focused jump-and-repair constraint handling for fixed-parameter tractable graph problems closed under induced subgraphs. *Theoretical Computer Science*, 951:113719, 2023.

Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. In Benjamin Doerr and Frank Neumann (eds.), *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pp. 1–87. Springer, 2020.

Benjamin Doerr, Yassine Ghannane, and Marouane Ibn Brahim. Runtime analysis for permutation-based evolutionary algorithms. *Algorithmica*, 86(1):90–129, 2024.

Arkadiy Dushatskiy, Marco Virgolin, Anton Bouter, Dirk Thierens, and Peter A. N. Bosman. Parameterless gene-pool optimal mixing evolutionary algorithms. *Evolutionary Computation*, 32 (4):371–397, 12 2024.

Brian W. Goldman and William F. Punch. Parameter-less population pyramid. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '14)*, pp. 785–792. ACM, 2014.

Brian W. Goldman and Dirk Sudholt. Runtime analysis for the parameter-less population pyramid. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '16)*, pp. 669–676. ACM, 2016.

Hongmei He and Ondrej Sýkora. New circular drawing algorithms. *Proc. of ITAT*, 4:15–19, 2004.

Shih-Huan Hsu and Tian-Li Yu. Optimization by pairwise linkage detection, incremental linkage set, and restricted / back mixing: DSMGA-II. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '15)*, pp. 519–526. ACM, 2015.

Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger (eds.), *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, pp. 85–103, Boston, MA, 1972. Springer US.

Lázaro Lugo, Carlos Segura, and Gara Miranda. A diversity-aware memetic algorithm for the linear ordering problem. *Memetic Computing*, 14(4):395–409, 2022.

Christian Matuszewski, Robby Schönfeld, and Paul Molitor. Using sifting for k-layer straightline crossing minimization. In *Proceedings of Graph Drawing: 7th International Symposium, GD'99*, pp. 217–224. Springer, 1999.

Masaharu Munetomo and David E. Goldberg. Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation*, 7(4):377–398, 1999.

Matthew Newton, Ondrej Sýkora, Mark Withall, and Imrich Vrt'o. A parallel approach to row-based VLSI layout using stochastic hill-climbing. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pp. 750–758. Springer, 2003.

Andre Opris, Sebastian Sonntag, and Dirk Sudholt. A royal road function for permutation spaces: an example where order crossover is provably essential. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '25)*, pp. 1631–1640. ACM, 2025.

Michal W. Przewozniczek and Marcin M. Komarnicki. Empirical linkage learning. *IEEE Transactions on Evolutionary Computation*, 24(6):1097–1111, 2020.

Michal W Przewozniczek, Marcin M Komarnicki, Peter AN Bosman, Dirk Thierens, Bartosz Frej, and Ngoc Hoang Luong. Hybrid linkage learning for permutation optimization with gene-pool optimal mixing evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '21)*, pp. 1442–1450. ACM, 2021a.

Michal W. Przewozniczek, Marcin M. Komarnicki, and Bartosz Frej. Direct linkage discovery with empirical linkage learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '21)*, pp. 609–617. ACM, 2021b.

Michal Witold Przewozniczek, Renato Tinós, and Marcin Michal Komarnicki. First improvement hill climber with linkage learning – on introducing dark gray-box optimization into statistical linkage learning genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '23)*, pp. 946–954. ACM, 2023.

Yukai Qiao and Marcus Gallagher. Analyzing the runtime of the gene-pool optimal mixing evolutionary algorithm (GOMEA) on the concatenated trap function. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '24)*, pp. 1520–1526. ACM, 2024.

Marcus Schmidbauer and Dirk Sudholt. Empirical linkage learning provably builds truthful models on concatenated traps and H-IFF. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '25)*, pp. 827–835. ACM, 2025.

Carlos Segura, Lázaro Lugo, Gara Miranda, and Edison David Serrano Cárdenas. PACE 2024 heuristic track submission. `https://github.com/carlossegurag/PaceChallenge24`, 2024. Accessed: 2025-01-14.

Michael Simpson, Venkatesh Srinivasan, and Alex Thomo. Efficient computation of feedback arc set at web-scale. *Proceedings of the VLDB Endowment*, 10:133–144, 2016.

Dirk Sudholt. Analysing the robustness of evolutionary algorithms to noise: refined runtime bounds and an example where noise is beneficial. *Algorithmica*, 83(4):976–1011, 2021.

Dirk Thierens and Peter A.N. Bosman. Optimal mixing evolutionary algorithms. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO '11)*, pp. 617–624. ACM, 2011.

Renato Tinós, Darrell Whitley, and Francisco Chicano. Partition crossover for Pseudo-Boolean optimization. In *Proceedings of the Conference on Foundations of Genetic Algorithms (FOGA '15)*, pp. 137–149. ACM, 2015.

Renato Tinós, Michal W. Przewozniczek, and Darrell Whitley. Iterated local search with perturbation based on variables interaction for pseudo-boolean optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '22)*, pp. 296–304. ACM, 2022.

Renato Tinós, Michal W. Przewozniczek, Darrell Whitley, and Francisco Chicano. Iterated local search with linkage learning. *ACM Transactions on Evolutionary Learning and Optimization*, 4 (2), 2024.

Darrell Whitley. *Next Generation Genetic Algorithms: A User's Guide and Tutorial*, pp. 245–274. Springer International Publishing, Cham, 2019a.

Darrell Whitley. A gray box manifesto for evolutionary combinatorial optimization. *SIGEVOlution*, 12(1):3–5, 2019b.

A Full Proof of Theorem 3.1

Theorem 3.1. *Let G be a DAG containing all transitive edges, with n vertices and m edges. Then, RLS optimises G in an expected number of $O(n^2 m)$ generations.*

Proof of Theorem 3.1. Let $\pi^{(t)}$ be the current permutation. Consider an arc $(u, v) \in E$ that is a shortest back arc in $\pi^{(t)}$, i.e., $v <_t u$ and u, v are closest with this property. Such an arc exists as long as the optimum is not found. For any vertex w with $v <_t w <_t u$ we show that w is connected to neither u nor v. Since (u, v) is a shortest back arc, it immediately follows that $(u, w), (w, v) \notin E$. Now, if $(w, u) \in E$, then transitivity and $(u, v) \in E$ imply $(w, v) \in E$, which is a contradiction. We can exclude $(v, w) \in E$ in a similar fashion.

Thus, u, v are not connected to any vertex between them and the fitness-change of jumping u to v (or vice versa) is positive, since the only arc that changes direction by such a jump is (u, v). Performing such a jump has probability $\Omega(1/n^2)$ in every generation and improves the fitness by 1. Since there are at most m back arcs in a permutation, an optimum is found after $O(n^2 m)$ steps in expectation. □

B Full Proof of the Lower Bound without Linkage Learning (Theorem 3.2)

In this appendix, we give a full proof of the weakly exponential lower bound for RLS without linkage learning on the double-star graph D_n. A roadmap for the proof is as follows. We first describe precisely which jumps are accepted by RLS and which are not (Lemma B.1). The special vertices r and ℓ may be chosen to jump, but they are also frequently moved by ± 1 position if an element jumps over r (ℓ) and thus r (ℓ) is shifted by one position (see Lemma B.2).

We then consider blocks of contiguous elements in the permutation and show that, if a block is contained between r and ℓ for a sufficient amount of time, it typically contains a mixture of red and blue vertices. The reason is that all jumps of elements in $L \cup R$ into such a block are accepted, regardless of the colour of the incoming element, and we have equal probabilities of choosing red and blue vertices, respectively. The reason for considering blocks is to control jumps of r and ℓ. Since in every global optimum, r and ℓ are consecutive, we want to show that they typically maintain a large distance between them. Large inward jumps of r are rejected when skipping red vertices, and the same holds symmetrically for inward jumps of ℓ when skipping blue vertices. Thus, when a block contains both red and blue vertices, it prevents jumps of r and ℓ that would skip the whole block.

The challenge is that in some situations, blocks between r and ℓ may contain only vertices of one colour. This happens, for example, when r performs a large outward jump, skipping over vertices that are all blue, and thus leaving behind blocks that are now contained between r and ℓ, but only contain vertices of a single colour. We address this by showing that blocks contained between r and ℓ become mixed in a short amount of time, owing to an influx of vertices from both colours (Lemma B.3).

Then, we show that there is a decent probability of creating a permutation in which r and ℓ occupy positions amongst the first and the last ξn positions, respectively, and all blocks in between have been subjected to mixing for a sufficient amount of time (Lemma B.4). We call such an event a *fallback* and define *periods* of time during which a fallback has a decent probability of occurring. We then analyse a period in detail and show that there are three possible outcomes: a fallback, a jump decreasing the distance between r and ℓ without skipping a whole block, and none of these two events. In the last two cases, we identify statements that allow us to iterate these considerations in another, independent period (Lemma B.5). Finally, we prove that, to find the optimum, we have to experience many periods that do not end in a fallback (or witness some other, rare events, details of which will be provided in the proof of Theorem 3.2). The expected waiting time for any such rare event establishes the claimed weakly exponential lower bound (Theorem 3.2).

We start by describing in the following lemma which jumps are accepted from a local optimum.

Lemma B.1 (Accepted jumps)**.** *For every permutation π with $r < \ell$ and the only back arc being (ℓ, r), the following holds.*

- *A jump of an element $v \in L$ to a position j is accepted iff $j < \pi_\ell^{-1}$.*
- *A jump of an element $v \in R$ to a position j is accepted iff $j > \pi_r^{-1}$.*
- *A jump of r to a smaller position $j \leq \pi_r^{-1}$ is accepted. A jump of r to a larger position $j \in \{\pi_r^{-1}+1, \ldots, \pi_\ell^{-1}-1\}$ is accepted iff π does not contain any elements in R at positions in $\{\pi_r^{-1}, \ldots, j\}$. Jumping r to position $j = \pi_\ell^{-1}$ is accepted iff there is at most one element in R at positions in $\{\pi_r^{-1}, \ldots, \pi_\ell^{-1}\}$. Jumping r to position $j = \pi_\ell^{-1}+1$ is accepted iff there is no element in R at positions in $\{\pi_r^{-1}, \ldots, \pi_\ell^{-1}\}$. Jumps of r to positions $j > \pi_\ell^{-1}+1$ are always rejected.*
- *A jump of ℓ to a larger position $j \geq \pi_\ell^{-1}$ is accepted. A jump of ℓ to a smaller position $j \in \{\pi_r^{-1}+1, \ldots, \pi_\ell^{-1}-1\}$ is accepted iff π does not contain any elements in L at positions in $\{j, \ldots, \pi_\ell^{-1}\}$. Jumping ℓ to position $j = \pi_r^{-1}$ is accepted iff there is at most one element in L at positions in $\{\pi_r^{-1}, \ldots, \pi_\ell^{-1}\}$. Jumping ℓ to position $j = \pi_r^{-1}-1$ is accepted iff there is no element in L at positions in $\{\pi_r^{-1}, \ldots, \pi_\ell^{-1}\}$. Jumps of ℓ to positions $j < \pi_r^{-1}-1$ are always rejected.*

Proof. If $v \in L$ is jumped to a position $j < \pi_\ell^{-1}$ then the arc (v, ℓ) remains forward in the offspring, thus the fitness is unchanged. If it is jumped to a position $j \geq \pi_\ell^{-1}$, the arc (v, ℓ) becomes a back arc, and the offspring is rejected. The second statement is analogous.

A jump of r to a smaller position $j \leq \pi_r^{-1}$ does not change the orientation of any arc since for all permutations where (ℓ, r) is the only back arc, all vertices preceding r are from L. A jump of r to a larger position $\pi_r^{-1} < j < \pi_\ell^{-1}$ is accepted if π does not contain any elements in R at positions in $\{\pi_r^{-1}, \ldots, j\}$, since in this case no arc orientation is altered. Otherwise, if there is an element $v \in R$ skipped by r then the arc (v, r) becomes a back arc, and the offspring is rejected. If r jumps past ℓ, the arc (ℓ, r) becomes a forward arc. If no vertex in R is skipped, a global optimum is found. If at most one vertex in R is skipped, the jump is accepted. This proves the statement about jumping r to position π_ℓ^{-1}. A jump to $j = {\pi^{(t)}}_\ell^{-1} + 1$ is accepted iff there is no element in R at positions in $\{\pi_r^{-1}, \ldots, \pi_\ell^{-1}\}$. Since all vertices to the right of ℓ are from R, any jump to $j \geq {\pi^{(t)}}_\ell^{-1} + 2$ is rejected as then r skips at least two vertices in R.

The fourth statement is analogous. □

The following lemma states probabilities for r and ℓ to be shifted by one position to the left and to the right, respectively.

Lemma B.2 (Probabilities of shifts)**.** *For every permutation π with $r < \ell$ and the only back arc being (ℓ, r), the following holds. Define $i_r \coloneqq \pi_r^{-1}$ and $i_\ell \coloneqq \pi_\ell^{-1}$ as the positions of r and ℓ in π, respectively. We say that r (ℓ) is shifted if its position is changed by ± 1 owing to an accepted jump of an element in $L \cup R$. The probabilities of shifts in $\pi^{(t)}$ are as follows.*

$$\Pr(\textit{left-shift of } r) = \frac{(i_r - 1)(i_\ell - i_r)}{n(n-1)}$$

$$\Pr(\textit{right-shift of } r) = \frac{(n/2 - i_r)i_r}{n(n-1)} \leq \frac{i_r/2}{n-1}$$

$$\Pr(\textit{right-shift of } \ell) = \frac{(n - i_\ell)(i_\ell - i_r)}{n(n-1)}$$

$$\Pr(\textit{left-shift of } \ell) = \frac{(i_\ell - n/2 - 1)(n - i_\ell + 1)}{n(n-1)} \leq \frac{(n - i_\ell + 1)/2}{n-1}.$$

Proof. Node r is shifted one position to the left iff one of the $i_r - 1$ elements left of r is jumped to one of the $i_\ell - i_r$ indices in $\{i_r, \ldots, i_\ell - 1\}$. Node ℓ is shifted one position to the right iff one of the

$n - i_\ell$ elements right of ℓ is jumped to one of the $i_\ell - i_r$ indices in $\{i_r + 1, \dots i_\ell\}$. Thus,

$$\Pr(\text{left-shift of } r) = \frac{(i_r - 1)(i_\ell - i_r)}{n(n-1)}$$
$$\Pr(\text{right-shift of } \ell) = \frac{(n - i_\ell)(i_\ell - i_r)}{n(n-1)}.$$

Node r is shifted one position to the right iff an element from L between r and ℓ is jumped to a position in $\{1, \dots, i_r\}$. Node ℓ is shifted one position to the left iff an element from R between r and ℓ is jumped to a position in $\{i_\ell, \dots, n\}$. Since there are no elements in L to the right of ℓ, there are $|L| - (i_r - 1) = n/2 - i_r$ elements from L between r and ℓ. Likewise, there are $|R| - (n - i_\ell) = n/2 - n + i_\ell - 1 = i_\ell - n/2 - 1$ elements from R between r and ℓ. Together,

$$\Pr(\text{right-shift of } r) = \frac{(n/2 - i_r)i_r}{n(n-1)} \leq \frac{i_r/2}{n-1}$$
$$\Pr(\text{left-shift of } \ell) = \frac{(i_\ell - n/2 - 1)(n - i_\ell + 1)}{n(n-1)} \leq \frac{(n - i_\ell + 1)/2}{n-1}. \qquad \square$$

In the following analysis, we consider *blocks* of elements. A block is defined as an interval of positions. Note that the elements contained in this block depend on the current permutation. In particular, $\pi^{(t+1)}$ may contain different elements at positions in S compared to $\pi^{(t)}$. This even holds if jumps do not affect the block directly, as elements at positions in S may be shifted left or right by one position if an element jumps over the block. Hence, while the positions in S remain fixed over time, the elements currently located at those positions may change. We define the size of blocks considered in the following and ignore rounding issues for the sake of readability.

Definition 1. *Let $k \coloneqq 3\tau$ be the block size, and $\alpha \coloneqq \tau k/(16n) = 3n^{1/3}/16$. A block of size k is called* mixed *if it contains at least α elements from R and at least α elements from L. Let $\tau \coloneqq n^{2/3}$ denote the time allowed for a block to mix.*

Blocks in between r and ℓ typically contain elements from both R and L. However, this is not always the case. Recall that in every current search point, only elements from L may be placed left of r. So, imagine a current permutation starting with a long sequence of vertices from L. If then r makes a jump of reasonable size to the left, it skips over a region of vertices in L. Thus, in the resulting permutation, there will be a region to the *right* of r that contains only vertices in L. Consequently, a block in this region only contains elements from one set. However, if the block has evolved for some time during which it was contained within r and ℓ, it typically contains a mixture of elements from R and L. The following notion captures this.

Definition 2. *Let $t \geq 0$ and $a, b \in \{1, \dots, n\}$ with $a \leq b$. An interval $[a, b]$ is called* established *with respect to a current permutation $\pi^{(t)}$ if it has been fully contained within r and ℓ for at least $\min\{t, \tau\}$ steps:*

$$\forall t' \in [\max\{t - \tau + 1, 0\}, t] \colon \{{\pi^{(t')}}_r^{-1} < a\} \wedge \{b < {\pi^{(t')}}_\ell^{-1}\}.$$

Lemma B.3. *Consider a block S that is fully contained between r and ℓ in the current permutation $\pi^{(t)}$, i.e., if i is the first position in S, we have $\{{\pi^{(t)}}_r^{-1} < i\} \wedge \{i + k - 1 < {\pi^{(t')}}_\ell^{-1}\}$. Then,*

$$\Pr(\{S \text{ is not established at time } t + \tau\} \cup \{S \text{ is mixed in } \pi^{(t+\tau)}\}) \geq 1 - e^{-\Omega(n^{1/3})}.$$

Moreover, for the first τ generations, that is, for all $t \in [0, \tau]$, we have

$$\Pr(\{S \text{ is not established at time } t\} \cup \{S \text{ is mixed in } \pi^{(t)}\}) \geq 1 - e^{-\Omega(n^{1/3})}.$$

Note that, when conditioning on S being established at time $t + \tau$, then the above event simplifies to "S is mixed in $\pi^{(t+\tau)}$".

Proof of Lemma B.3. We use the following inequality for two events B, C,

$$\begin{aligned}\Pr(B \cup C) &= \Pr(B \cup C \mid B)\Pr(B) + \Pr(B \cup C \mid \overline{B})\Pr(\overline{B}) \\ &= \Pr(B) + \Pr(C \mid \overline{B})(1 - \Pr(B)) \geq \Pr(C \mid \overline{B}).\end{aligned}$$

Hence, it suffices to compute a lower bound on the probability of $\{S \text{ is mixed in } \pi^{(t+\tau)}\}$ (which is event C) while assuming $\{S \text{ is established at time } t+\tau\}$ (which is event $\overline{B}$).

We first show the statement assuming no conditions on whether r and/or ℓ may jump. Then every element is selected to jump with probability $1/n$. As long as S is sandwiched between r and ℓ, every jump of an element in $V \setminus \{r, \ell\}$ to any index in S is accepted. We consider the subset $S' \subseteq S$ that excludes the first and the last τ positions in S and note that if an element jumps into S' and is not affected by any further jump in the period, it will still be contained in S at the end of the τ generations, since it can only be shifted at most τ times and thus cannot leave S.

The expected number of red elements jumping to any index in S' is

$$\tau \cdot (k - 2\tau) \cdot (n/2 - 1)/n^2 \geq \frac{\tau k}{6n}(1 - O(1/n)).$$

By Chernoff bounds, this is at least $2\alpha \coloneqq \tau k/(8n)$ with probability $1 - e^{-\Omega(\alpha)}$. We assume that this happens, fix 2α of these red elements and call them *new*. Note that new elements may be removed from S before $\pi^{(t+\tau)}$ is created, and so we need to show that at least α new elements are still present in $\pi^{(t+\tau)}$.

A necessary condition for removing a new element is to jump it outside of S'. This is possible, in principle, for all generations $t+1, \dots, t+\tau$. We bound the probability that at least α elements are removed within τ generations. To obtain an upper bound, we pessimistically assume that all α elements which we want to remove are already present at time t. In the actual process, new elements appear only later, which reduces the number of opportunities for their removal. Consequently, our combinatorial argument counts many removal sequences that cannot occur in the true process, and therefore yields a valid (though loose) upper bound on the desired probability.

A necessary condition for removing at least α elements is to have a subset of α elements out of all 2α new elements, and all elements of this subset jump outside of S'. There are $\binom{2\alpha}{\alpha} \leq 4^\alpha$ such subsets. There are $\binom{\tau}{\alpha}$ ways of choosing a generation in which to jump a new element, and there are $\alpha!$ orderings of these elements. The probability of jumping all α elements in these fixed times, and in a fixed ordering, is at most $(1/n)^\alpha$. Together, the probability of removing any selection of α new red elements is at most

$$\binom{2\alpha}{\alpha}\binom{\tau}{\alpha}\alpha!\left(\frac{1}{n}\right)^\alpha \leq \left(\frac{4\tau}{n}\right)^\alpha = e^{-\Omega(\alpha)}$$

with room to spare. The same arguments hold for blue vertices. Taking a union bound over all failure events from both colours, at least α new red elements and at least α new blue elements remain with probability at least $1 - 4e^{-\Omega(\alpha)}$. Noting that $4e^{-\Omega(\alpha)} = e^{-\Omega(\alpha)+O(1)} = e^{-\Omega(n^{1/3})}$ completes the proof.

For the final statement, we assume that S is established at time t as otherwise there is nothing to prove. In the initial permutation, S is populated from vertices in $L \cup R$ chosen uniformly at random without replacement. We claim that the probability of S containing fewer than 2α vertices from R is at most $e^{-\Omega(\alpha)}$. Defining indicator variables $X_1, \dots, X_k$ where $X_i = 1$ iff the i-th element in S is from R, these indicator variables are negatively correlated, and standard Chernoff bounds apply (see, e. g. Sections 1.10.2.2 and 1.10.2.3 in Doerr (2020)). Now Chernoff bounds imply that the probability of having fewer than 2α elements from R in S is at most $e^{-\Omega(\alpha)}$, and a symmetric statement applies to elements from L. Assuming S contains at least 2α elements from L and at least 2α elements from R, we call 2α vertices of each colour *new* and continue the train of thought from above. □

Definition 3. *Let $\xi \coloneqq 1/16$. A permutation π is called* safe *if r is at a position at most ξn, ℓ is at a position at least $n - \xi n$, and all blocks fully contained in $[\xi n, n - \xi n]$ are established in π.*

The following lemma shows that the algorithm can easily fall back to a safe permutation within 2τ generations. This time span allows for r and ℓ to jump to appropriate positions during the first τ generations, and to arrive at a state where all mentioned blocks are mixed after further τ generations.

Lemma B.4 (Fallbacks)**.** *For any current permutation $\pi^{(t)}$, RLS reaches a safe permutation within the next 2τ generations with probability $\Omega(\tau^2/n^2) = \Omega(n^{-2/3})$.*

Proof. The probability of r jumping to a position at most $\xi n - 2\tau \geq \xi n/2$ in some fixed generation $t_r \in [t, t+\tau-1]$ is $\xi n/(2n^2) = \xi/(2n)$. The same holds for the probability of ℓ jumping to a position at least $n - \xi n + 2\tau$ in some fixed generation t_ℓ. The probability of not selecting r for another jump at all times in $[t_r+1, t+2\tau-1]$, and not selecting ℓ for another jump at all times in $[t_\ell+1, t+2\tau-1]$ is always at least $(1-2/n)^{2\tau} \geq 1 - 4\tau/n = \Omega(1)$ by Bernoulli's inequality, irrespective of t_r and t_ℓ. As there are $\binom{\tau}{2}$ possible choices of t_r and t_ℓ, the probability of all mentioned events happening is at least

$$\binom{\tau}{2}\left(\frac{\xi}{2n}\right)^2\left(1-\frac{2}{n}\right)^{2\tau} = \Omega(\tau^2/n^2).$$

Note that if these events occur, shifts can only move r and ℓ by a total distance of at most 2τ, respectively, after jumping them the first time. Hence, r and ℓ will remain outside of the interval $[\xi n, n-\xi n]$ at all times in $[t+\tau, t+2\tau]$, rendering this interval established. Thus, the probability that RLS reaches a safe permutation within the next 2τ generations is $\Omega(\tau^2/n^2)(1-o(1)) = \Omega(\tau^2/n^2)$. □

We define the distance $\Delta^{(t)} := {\pi^{(t)}}_\ell^{-1} - {\pi^{(t)}}_r^{-1}$ as the distance between the positions of ℓ and r in the current permutation at time t.

We will consider *periods* of up to 2τ generations, at the start of which there is an established interval $[a_0, b_0]$ of maximum size. During the period, there is a chance to experience a fallback according to Lemma B.4. A period may end prematurely if the algorithm performs a jump of r or ℓ to a location close to a_0 or b_0, respectively. The latter event is denoted G, and it implies that the maximum length of any established interval may shrink. We still show that, if this happens, the maximum interval length does not shrink by too much. If none of the events happens, we show how to identify an established interval at the end of the period that is at least as large as the initial one $[a_0, b_0]$, so that we can invoke another call of Lemma B.5. The following lemma is at the heart of our proof.

Lemma B.5. *Consider a permutation $\pi^{(t_0)}$ with $\Delta^{(t_0)} \geq (4/5)n$. Let $a_0 < b_0$ such that $[a_0, b_0]$ is the largest established interval at time t_0.*

Let $G^{(t)}$ denote the event that during the creation of $\pi^{(t)}$ either r is selected to jump and the destination is chosen in $[a_0 - 2\tau, a_0 + 2\tau + k - 1]$ or ℓ is selected to jump and the destination is chosen in $[b_0 - 2\tau - k + 1, b_0 + 2\tau]$. Let $G := \bigcup_{t=t_0+1}^{t_0+2\tau} G^{(t)}$ and $t_G = \inf\{t \mid t > t_0, G^{(t)}\}$ denote the first time this has happened, if this event occurs.

Consider a period of steps starting at time t_0 and ending at time $t_{\text{end}} := \min\{t_G, t_0 + 2\tau\}$. Then the following statements hold.

(1) $\Pr(G) = O(n^{-2/3})$.

(2) With probability $1 - e^{-\Omega(n^{1/3})}$, at the end of the period, $[a_0 + 2\tau + k, b_0 - 2\tau - k]$ is established in $\pi^{(t_{\text{end}})}$.

(3) If $\Delta^{(t_0)} \in [(4/5)n + 2\tau, (19/20)n - 2\tau]$ then, with probability $1 - e^{-\Omega(n^{1/3})}$, event G occurs or at time $t_0 + 2\tau$, there is an interval $[a_{\text{end}}, b_{\text{end}}]$ of length $b_{\text{end}} - a_{\text{end}} + 1 \geq b_0 - a_0 + 1$ that is established in $\pi^{(t_0+2\tau)}$.

(4) If $\Delta^{(t_0)} \geq (19/20)n - 2\tau$ then, with probability $1 - O(n^{-1/3})$, a safe permutation is found within the next 2τ generations.

Proof. **Proof of Statement (1).** By definition of $G^{(t)}$, for the event to occur, we must choose r or ℓ to jump and choose one of $4\tau + k$ destinations. The probability of choosing r or ℓ is $2/n$ and the probability of choosing a suitable destination is at most $(4\tau + k)/(n-1) = O(n^{-1/3})$. Together, $\Pr(G^{(t)}) = O(n^{-4/3})$ and a union bound shows that $\Pr(G) \leq \sum_{t=t_0+1}^{t_0+2\tau} \Pr(G^{(t)}) = 2\tau \cdot O(n^{-4/3}) = O(n^{-2/3})$.

Proof of Statement (2). Let $S_r := [a_0 + 2\tau, a_0 + 2\tau + k - 1]$ be a block inside $[a_0, b_0]$, that is, at least 2τ to the right of the position of r in $\pi^{(t_0)}$. We remark that this distance of 2τ will be used as a buffer to account for shifts of r during at most 2τ generations. As long as S_r remains fully

contained between r and ℓ, S_r remains established. Hence, if S_r remains fully contained for the next 2τ iterations, S_r is established at all times $t+\tau \in \{t_0+1, \dots, t_0+2\tau\}$. Applying Lemma B.3 repeatedly for all such t and applying a union bound shows that S_r remains mixed during the next 2τ generations with probability $1 - 2\tau \cdot e^{-\Omega(n^{1/3})} = 1 - e^{-\Omega(n^{1/3})}$. Consequently, any jump of r from a position at most $a_0 + 2\tau - 1$ (that is, from the left of S_r) to any position at least $a_0 + 2\tau + k$ that is, skipping all elements of S_r), will be rejected.

Using the fact that a jump of r skipping S_r is always rejected, we show by induction over j that at time $t_0 + j$, the position of r in $\pi^{(t_0+j)}$ is at most $a_0 + j - 1$, unless G occurs. The claim holds for $j = 0$ since $[a_0, b_0]$ is established at time t_0 and thus $r_{t_0}^{-1} \le a_0 - 1$. If at time $t_0 + j$, r is not jumped, the claim holds since r can only be shifted to the right by 1. Thus, if at time $t_0 + j$ it is at a position at most $a_0 + j - 1$, it will be at a position at most $a_0 + j$ at time $t_0 + j + 1$. If r is being jumped at time $t_0 + j$, we distinguish the following cases. If the destination is at least $a_0 + 2\tau + k$, r would skip the established block S_r, and the step is rejected. Thus, r remains at a position at most $a_0 + j - 1$. If the destination is chosen in $[a_0 - 2\tau, a_0 + 2\tau + k]$, the event G is triggered, and the statement holds at time $t_0 + j + 1$. If the destination is at most $a_0 - 2\tau - 1$, the statement clearly holds at time $t_0 + j + 1$.

Consequently, r will remain at a position at most $a_0 + 2\tau - 1$ throughout the period, unless G occurs. Moreover, if G is triggered (at a time we introduced as t_G) by a jump of r, r will jump to a position at most $a_0 + 2\tau + k - 1$ in the offspring by definition of G. The offspring may be accepted or rejected, and in both cases, the period ends, and so r does not enter the interval $[a_0 + 2\tau + k, b_0 - 2\tau - k]$ during the period. All the above arguments hold symmetrically for jumps of ℓ and the block $S_\ell := [b_0 - 2\tau - k + 1, b_0 - 2\tau]$. Taking a union bound over failure probabilities $e^{-\Omega(n^{1/3})}$ from applications of Lemma B.3 for both considering r and ℓ, the second statement follows.

Proof of Statement (3). Under the same assumptions as for statement (2), as shown before, only jumps of r to a position at most $a_0 - 2\tau - 1$ are accepted, and the same holds symmetrically for jumps of ℓ.

We first work under the assumption that no such jumps of r and ℓ occur, and later explain how such jumps can be included in our analysis. Without accepted jumps of r and ℓ, only one of these vertices can move by ± 1 in any step. Consequently, the bounds on $\Delta^{(t_0)}$ imply that, for all $t \in [t_0, t_0 + 2\tau]$, $\Delta^{(t)} \in [(4/5)n, (19/20)n]$. We then have the following probability bounds on the motion of r and ℓ that hold universally throughout the period. We shall use shorthands $r_t := {\pi^{(t)}}_r^{-1}$ and $\ell_t := {\pi^{(t)}}_\ell^{-1}$ for the positions of r and ℓ in $\pi^{(t)}$, respectively, and note that then $\Delta^{(t)} = \ell_t - r_t$.

We assume without loss of generality that r is closer to its respective end of the permutation at time t_0, that is, $r_{t_0} \le n - \ell_{t_0}$. Then throughout the period, that is, for all $t \in \{t_0, \dots, t_0 + 2\tau\}$, the weaker bound $r_t \le n - \ell_t + 2\tau$ applies. Note that this assumption introduces an asymmetry in the way we handle r and ℓ. We first focus on the movement of ℓ and show that there is a bias towards moving ℓ outward.

By Lemma B.2, we have

$$\begin{aligned}\Pr(\text{right-shift of } \ell) - \Pr(\text{left-shift of } \ell) &= \frac{(n-\ell_t)(\ell_t - r_t)}{n(n-1)} - \frac{(\ell_t - n/2 - 1)(n - \ell_t + 1)}{n(n-1)} \\ &= \frac{(n-\ell_t)(n/2 + 1 - r_t) - (\ell_t - n/2 - 1)}{n(n-1)}\end{aligned}$$

From $\Delta^{(t)} = \ell_t - r_t \le (19/20)n$ and $r_t \le n - \ell_t + 2\tau$ it follows that $\ell_t \le (39/40)n + 2\tau$, or, equivalently, $n - \ell_t \ge n/40 - 2\tau$. Along with $r_t \le n - \Delta^{(t)} \le n/5$, implying $(n/2 + 1 - r_t) \ge (3/10)n$, and the trivial bound $\ell_t - n/2 - 1 \le n/2$, we get

$$\Pr(\text{right-shift of } \ell) - \Pr(\text{left-shift of } \ell) \ge \frac{(n/40 - 2\tau)(3/10)n - n/2}{n(n-1)} \ge \frac{1}{150} \tag{2}$$

if n is large enough.

At the start of the period, $r_{t_0} \le a_0 - 1$ and $\ell_{t_0} \ge b_0 + 1$ since $[a_0, b_0]$ is established in $\pi^{(t_0)}$. We now aim to show that, with high probability, ℓ reaches a position of at least $b_0 + c_1\tau + 1$, where $c_1 > 0$

is some constant, in the first τ steps of the period, that is, $\ell_{t_0+\tau} \geq b_0 + c_1\tau + 1$ and that during the following τ steps of the period, ℓ remains at positions at least $b_0 + c_2\tau + 1$ for some constant $0 < c_2 < c_1$; in other words, at all times $t \in [t_0 + \tau + 1, t_0 + 2\tau]$, we have $\ell_t \geq b_0 + c_2\tau + 1$.

Since the process moves only by $\{-1, 0, +1\}$ at each step and the probability bound from (2) holds conditionally for every step during the period given any historical context, we may apply the Azuma–Hoeffding inequality (Theorem 1.10.30 in Doerr (2020)) to a suitable martingale process as follows.

Let $X_{t_0+1}, \ldots, X_{t_0+\tau}$ denote the shifts of ℓ at each time step, where $X_t \in \{-1, 0, +1\}$. By (2), we have $\mathbb{E}(X_t \mid \mathcal{F}_{t-1}) \geq 1/150$ for all t, where $\mathcal{F}_{t-1}$ is the natural filtration up to step $t-1$. For $t \in \{t_0, \ldots, t_0 + \tau\}$, define the process

$$M_t \coloneqq \sum_{j=t_0+1}^{t} \Big(X_j - \mathbb{E}(X_j \mid \mathcal{F}_{j-1}) \Big) \quad \text{with } M_{t_0} = 0.$$

By construction, $(M_t)_{t=t_0}^{t_0+\tau}$ is a martingale with respect to $(\mathcal{F}_t)_{t=t_0}^{t_0+\tau}$. Furthermore, since $X_t \in \{-1, 0, +1\}$ and $1/150 \leq \mathbb{E}(X_t \mid \mathcal{F}_{t-1}) \leq 1$, the step differences satisfy $|M_t - M_{t-1}| = |X_t - \mathbb{E}(X_t \mid \mathcal{F}_{t-1})| \leq 2$ almost surely.

Applying the Azuma–Hoeffding inequality to (M_t), we obtain:

$$\Pr\left(\sum_{t=t_0+1}^{t_0+\tau} X_t < \frac{\tau}{200} \right) \leq \Pr\left(M_{t_0+\tau} < \frac{\tau}{200} - \frac{\tau}{150} \right) = \Pr\left(M_{t_0+\tau} < -\frac{\tau}{600} \right) \leq e^{-\Omega(\tau)}.$$

Hence, the total shift is at least $\tau/200$ with probability at least $1 - e^{-\Omega(\tau)} \geq 1 - e^{-\Omega(n^{1/3})}$ (generously bounding $\tau = \Theta(n^{2/3})$ from below by $\Omega(n^{1/3})$), which implies $\ell_{t_0+\tau} \geq b_0 + \tau/200 + 1$.

Assuming that this happens, we consider the τ steps starting at time $t_0 + \tau$ and note that we can apply the Azuma–Hoeffding inequality in the same way to random variables $X_{t_0+\tau}, \ldots, X_{t_0+2\tau-1}$. We consider partial sums $S_i \coloneqq \sum_{t=t_0+\tau+1}^{t_0+\tau+i} X_t$ and note that $\mathrm{E}(S_i) \geq 0$ for all $i \in \{1, \ldots, \tau\}$. We now invoke Theorem 1.10.31 in Doerr (2020), which gives bounds on the lower tail of partial sums:

$$\Pr(\exists i \in \{1, \ldots, \tau\} \colon S_i \leq -\tau/400) \leq \Pr(\exists i \in \{1, \ldots, \tau\} \colon S_i \leq \mathrm{E}(S_i) - \tau/400) \leq e^{-\Omega(\tau)},$$

where the right-hand side is the probability bound obtained from applying the Azuma-Hoeffding inequality to a martingale based on the whole sum S_τ. Rewriting the above and using $e^{-\Omega(\tau)} \leq e^{-\Omega(n^{1/3})}$,

$$\Pr(\exists t \in [t_0 + \tau + 1, t_0 + 2\tau] \colon \ell_t \leq \ell_{t_0+\tau} - \tau/400) \leq e^{-\Omega(n^{1/3})}.$$

This implies that, with probability $1 - e^{-\Omega(n^{1/3})}$, we have $\ell_t \geq \ell_{t_0+\tau} - \tau/400 \geq b_0 + \tau/400 + 1$ for all $t \in \{t_0 + \tau + 1, \ldots, t_0 + 2\tau\}$.

We argue similarly for the position of r, though we estimate its movement more conservatively since for small positions r_t, there is a weak drift towards outward moves. Consequently, we aim to show that r does not move inwards by too much, specifically $r_t \leq a_0 + \tau/400$ for all $t \in \{t_0 + 1, \ldots, t_0 + 2\tau\}$ (note the asymmetry to the derived statement on ℓ). Using Lemma B.2 and $\Delta^{(t)} \geq (4/5)n$,

$$\Pr(\text{left-shift of } r) = \frac{(r_t - 1)(\ell_t - r_t)}{n(n-1)} \geq \frac{(4/5)(r_t - 1)}{n-1}$$
$$\Pr(\text{right-shift of } r) = \frac{(n/2 - r_t)r_t}{n(n-1)} \leq \frac{r_t/2}{n-1}.$$

As long as $r_t \geq 3$, the expected total shift of r is

$$\Pr(\text{left-shift of } r) - \Pr(\text{right-shift of } r) \geq \frac{(4/5)(r_t - 1)}{n-1} - \frac{r_t/2}{n-1} \geq \frac{r_t/30}{n-1} \geq 0$$

and hence applying the Azuma–Hoeffding inequality and Theorem 1.10.31 in Doerr (2020) as before yields that

$$\Pr(\exists t \in [t_0 + 1, t_0 + 2\tau] \colon r_t \geq a_0 + \tau/400) \leq e^{-\Omega(n^{1/3})}.$$

This implies that, with probability $1 - e^{-\Omega(n^{1/3})}$, we have $r_t \leq a_0 - 1 + \tau/400$ for all $t \in \{t_0 + 1, \dots, t_0 + 2\tau\}$. Here, we assume that we always have $r_t \geq 3$. If this assumption is violated, we argue as follows. From $r_t = 2$ we need at least $\tau/400 - 1$ rightward shifts in at most 2τ steps, and each rightward shift of r with $r_t \leq \tau/400 - 1$ has a probability of at most $\tau/800/(n-1)$. The probability of seeing at least $\tau/400 - 1$ such rightward shifts in 2τ generations is at most, using $\binom{2\tau}{\tau/400-1} \leq 2^{2\tau} = (2^{800})^{\tau/400}$,

$$\binom{2\tau}{\tau/400-1} \cdot \left(\frac{\tau/800}{n-1}\right)^{\tau/400-1} \leq 2^{800} \left(\frac{2^{800} \cdot \tau/800}{n-1}\right)^{\tau/400-1} = e^{-\Omega(n^{1/3})} \tag{3}$$

with room to spare.

The preceding arguments assume that r does not jump to a position at most $a_0 - 2\tau - 1$ and ℓ does not jump to a position at least $b_0 + 2\tau + 1$. Note that a long outwards jump of r can be followed by a later long inward jump of r, and likewise for ℓ. Hence, it is not immediately obvious that a long outward jump is beneficial to the analysis. However, we have defined the event G such that any jump of r or ℓ leaves a gap of 2τ to $a_0 - 1$ and $b_0 + 1$, respectively. Note that this holds for all jumps of r and ℓ in case multiple such jumps occur in the same period of 2τ steps. Now it becomes apparent that such jumps are beneficial to the analysis. If a jump of ℓ to a position at least $b_0 + 2\tau + 1$ occurs within the first τ steps, the statement $\ell_{t_0+\tau} \geq b_0 + \tau/200 + 1$ follows deterministically. This is because following the last such jump before time $t_0 + \tau$, there are fewer than τ steps remaining until $t_0 + \tau$; even in the worst case, where ℓ is shifted inwards by 1 in each step, the bound holds. Assuming $\ell_{t_0+\tau} \geq b_0 + \tau/200$, if an outward jump of ℓ occurs during the following τ steps, $\ell_{t_0+2\tau} \geq b_0 + \tau/400 + 1$ also holds deterministically. Likewise, an outward jump of r during the period, along with less than 2τ inward shifts of r after the jump, implies that $r_{t_0+2\tau} \leq a_0+1 \leq a_0+\tau/400+1$ as before. Hence, the analysis also goes through when considering jumps of r and ℓ.

Using a union bound over all failure probabilities from Lemma B.3, all applications of Hoeffding bounds, and (3), with probability $1 - e^{-\Omega(n^{1/3})}$, the elements at positions in $[a_0 + \tau/400, b_0 + \tau/400] =: [a_{\text{end}}, b_{\text{end}}]$ will be contained in between r and ℓ for the last τ steps of the period, and hence the interval $[a_{\text{end}}, b_{\text{end}}]$ is established at time $t_0 + 2\tau$ and has the same size as $[a_0, b_0]$.

Proof of Statement (4). Recall $\xi = 1/16$. If n is large enough, we have $\Delta^{(t_0)} > (19/20)n - 2\tau \geq n - \xi n + 2\tau + 2$, which implies that $r_{t_0}^{-1} \leq \xi n - 2\tau - 2$ and $\ell_{t_0}^{-1} \geq n - \xi n + 2\tau + 2$. In this situation, if neither r nor ℓ is being chosen for a jump during the next τ steps, the interval $[\xi n, n - \xi n]$ is established at time $t_0 + \tau$ and the permutation $\pi^{(t_0+\tau)}$ is safe. The probability of this happening is at least $1 - 2\tau/n = 1 - O(n^{-1/3})$. □

Finally, we are in a position to prove the main result of this section, Theorem 3.2.

Proof of Theorem 3.2. With probability at least $1/n^2$, the initial permutation satisfies $r <_0 R \cup L <_0 \ell$, that is, the first element is r, and the last element is ℓ. We call this event A and will show $\mathrm{E}(T \mid A) \geq e^{\Omega(n^{1/3})}$, where T denotes the optimisation time of RLS. By the law of total probability, the expected optimisation time of RLS is at least

$$\mathrm{E}(T) = \mathrm{E}(T \mid A)\Pr(A) + \mathrm{E}(T \mid \overline{A})\Pr(\overline{A}) \geq e^{\Omega(n^{1/3})} \cdot \Omega(1/n^2) = e^{\Omega(n^{1/3}) - O(\log n)} = e^{\Omega(n^{1/3})}.$$

We assume that this happens and note that then the algorithm starts with a safe permutation.

We define a *trial* as a sequence of generations starting with a safe permutation and ending when another safe permutation is found, or when $\Delta^{(t)}$ has decreased below $(4/5)n$. In the latter case, we say that the trial is successful. Note that no global optimum can be found while $\Delta^{(t)} \geq (4/5)n$, as a jump operation swapping r and ℓ would skip $\Omega(n)$ vertices of the wrong colour, creating $\Omega(n)$ back arcs. We show that the probability of a successful trial is exponentially small. Since a successful trial is necessary to find a global optimum, and each trial lasts at least one generation, the number of trials until the first successful trial provides a lower bound on the optimisation time.

Let $L^{(t)}$ denote the length of the largest established interval in $\pi^{(t)}$ and note that $\Delta^{(t)} \geq L^{(t)}$. By the definition of safe permutations, at the start of a trial, denoted as time t_0, the interval $[\xi n, n - \xi n]$

is established. Since $\xi = 1/16$, at the start of a trial, we have $\Delta^{(t_0)} \geq L^{(t_0)} \geq n - 2\xi n = (7/8)n = (4/5)n + (3/40)n$, hence by $\Omega(n)$ larger than the distance bound $(4/5)n$. In particular, Lemma B.5 applies.

We distinguish between *medium-distance periods*, at the start of which we have $\Delta^{(t_0)} \leq (19/20)n - 2\tau$, and *high-distance periods*, where $\Delta^{(t_0)} > (19/20)n - 2\tau$. In every high-distance period, by Lemma B.5 (4), a safe permutation will be reached during the period with probability at least $p_{\text{high}} = 1 - O(n^{-1/3})$. The number of times a safe permutation is reached during m high-distance periods is stochastically dominated by a binomial distribution with parameters m and p_{high}. The probability of not reaching a safe permutation during m high-distance periods is thus at most $(1 - p_{\text{high}})^m$. At the end of a high-distance period, by Lemma B.5 (2), the interval $[a_0 + 2\tau + k, b_0 - 2\tau - k]$ is established at the end of the period. Since $L^{(t_0)} = b_0 - a_0 + 1$ (recall that $[a_0, b_0]$ is the maximum established interval at time t_0) and $[a_0 + 2\tau + k, b_0 - 2\tau - k]$ has a length of $b_0 - a_0 - 4\tau - 2k + 1 = L^{(t_0)} - 4\tau - 2k$, the maximum established interval at the end of the period has a length of at least $L^{(t_0)} - 4\tau - 2k$, and thus $L^{(t)}$ has reduced by at most $4\tau + 2k$ during the period. Choosing $m \coloneqq \lfloor (n/40)/(4\tau + 2k) \rfloor = \Theta(n^{1/3})$, the probability of not reaching a safe permutation is at most $(1 - p_{\text{high}})^{\lfloor (n/40)/(4\tau+2k) \rfloor} = (1 - p_{\text{high}})^{\Omega(n^{1/3})} \leq e^{-\Omega(n^{1/3})}$ (with room to spare, as $1 - p_{\text{high}} = o(1)$) and the total decrease of $L^{(t)}$ during these periods is at most $\lfloor (n/40)/(4\tau + 2k) \rfloor \cdot (4\tau + 2k) \leq n/40$.

In every medium-distance period, the algorithm may undergo a fallback, which generates a safe permutation and ends the trial. Let F denote the event that within 2τ generations from the start of the period, RLS reaches a safe permutation, and recall from Lemma B.4 that $\Pr(F) \geq p_F = \Omega(\tau^2/n^2)$ for a universal probability bound p_F. Let G denote the event from Lemma B.5 and recall that $\Pr(G) \leq p_G = O(\tau^2/n^2)$ by Lemma B.5 (1) for a universal probability bound p_G. In any period, the probability bounds p_F and p_G hold irrespective of what happened in previous periods. If neither F nor G occurs, according to Lemma B.5 (3), $L^{(t)}$ does not decrease, and we consider the next medium-distance phase until we encounter a medium-distance phase in which $F \cup G$ occurs. Now, the conditional probability of F occurring, given $F \cup G$, is at least

$$\frac{p_F}{p_F + p_G} \geq p_{\text{medium}}$$

for some constant $p_{\text{medium}} > 0$. Again, the probability of no safe permutation being found during $m = \lfloor (n/40)/(4\tau + 2k) \rfloor = \Theta(n^{1/3})$ medium-distance periods ending with $F \cup G$ is at most $(1 - p_{\text{medium}})^m = e^{-\Omega(n^{1/3})}$ and the total reduction of $L^{(t)}$ is at most $n/40$.

Note that even after deducting the aforementioned maximum reductions, we still have $\Delta^{(t)} \geq (7/8)n - n/40 - n/40 = (33/40)n > (4/5)n$, and the trial is not yet successful.

Finally, we sum up all failure probabilities by making extensive use of union bounds. Define $t_{\max} \coloneqq e^{cn^{1/3}}$ for a sufficiently small constant $c > 0$. Conditional on the initialisation stated before, the algorithm can only find a global optimum if one of the following failure events happens:

- There is a period in which one of the complementary events of those stated in Lemma B.5 occurs. Since there are $O(1)$ such events, each happening with a probability of at most $e^{-\Omega(n^{1/3})}$, a union bound results in a probability bound of $O(t_{\max}) \cdot e^{-\Omega(n^{1/3})}$.
- A trial contains m high-distance phases during which no safe permutation is discovered. As stated before, the probability of this event is $e^{-\Omega(n^{1/3})}$, and a union bound over at most $t_{\max}$ trials yields a probability bound of $t_{\max} \cdot e^{-\Omega(n^{1/3})}$.
- A trial contains m medium-distance phases ending in $F \cup G$ during which F never occurs. As stated before, the probability of this event is $e^{-\Omega(n^{1/3})}$, and a union bound over at most $t_{\max}$ trials yields a probability bound of $t_{\max} \cdot e^{-\Omega(n^{1/3})}$.

Now the sum of all failure probabilities is

$$O(t_{\max}) \cdot e^{-\Omega(n^{1/3})} = e^{cn^{1/3} + O(1)} \cdot e^{-\Omega(n^{1/3})} = e^{-\Omega(n^{1/3})}$$

if the constant c is chosen small enough. Taking into account the initialisation, the algorithm requires $t_{\max}$ generations with probability $\Omega(1/n^2)$ and thus the expected optimisation time is at least $\Omega(t_{\max}/n^2) = e^{cn^{1/3}-O(\log n)} = e^{\Omega(n^{1/3})}$. □

C Omitted proofs of Section 4

The linkage learning mechanism by Przewozniczek et al. (2021a) works as follows. Given a partition of $V = B \dot\cup C$ into two non-empty *blocks* B, C, Przewozniczek et al. (2021a) define B and C as *independent* if any two permutations π, ρ that induce the same order on B and on C have the same fitness. They consider $s, t \in V$ as linked if there is no partition of V into independent sets B, C with $s \in B$, $t \in C$.

Unfortunately, this definition turns out to be rather restrictive in our setting. Consider, for example, the graph $G = (V, E)$ with $V = \{s, u, v\}$ and $E = \{(s, u), (s, v)\}$. Intuitively, the relative order of u and v is irrelevant, and we would therefore like to regard them as not linked. However, we show in the following lemma that, for graphs that are connected after removing all bidirected arcs, every pair of vertices is linked according to the above definition.

Lemma C.1. *Let $G = (V, E)$ be a graph that is weakly connected after removing all bidirected arcs. Then there is no partition of V into two non-empty sets B and C that are independent.*

Proof. Assume for the sake of contradiction that B, C is such a partition. Since G is weakly connected even after removing all bidirected arcs, it follows that there exist vertices $u \in B$ and $v \in C$ that are connected by an arc e of G, which is not bidirected. Now consider any permutation π of V where u and v are consecutive and let π' be obtained by swapping u and v. Clearly, both π and π' induce the same orders on B and on C. However, $f(\pi) \neq f(\pi')$ as e is a forward arc in one of them and a back arc in the other one, while all other arcs are of the same type in both permutations. □

Lemma 4.1. *Let $G = (V, E)$ be an unweighted graph with adjacency matrix $A = (a_{u,v})_{u,v \in V}$ and let f be the objective function from Eq. (1). Let further π be a permutation of V, $u, v \in V$. We then have $d_{u,v}(\pi) = 1$ if and only if $a_{u,v} - a_{v,u} = 1$ and $\big[\sum_{w \in W_\pi^-(u,v)} (a_{w,u} - a_{u,w}) \in \{0, 1\}$ or $\sum_{w \in W_\pi^-(u,v)} (a_{w,v} - a_{v,w}) \in \{0, -1\} \big]$.*

Proof of Lemma 4.1. Recall that $W_\pi^-(u, v)$ denotes the elements strictly between u and v. For $x \in \{u, v\}$ let r_x (s_x) be the number of arcs from x to any w with $w \in W_\pi^-(u, v)$ (from any w with $w \in W_\pi^-(u, v)$ to x). Let $\alpha \coloneqq -\operatorname{sgn}(\pi_u^{-1} - \pi_v^{-1})$. Note that we have $\alpha = 1$ if $u <_\pi v$ and $\alpha = -1$ otherwise. Clearly, $s_x - r_x = \sum_{w \in W_\pi^-(u,v)} (a_{w,x} - a_{x,w})$ for $x \in \{u, v\}$. We further have

$$
\begin{aligned}
f(\pi_{u \to v}) &= f(\pi) + \alpha(r_u - s_u), \\
f(\pi_{u \leftarrow v}) &= f(\pi) + \alpha(s_v - r_v), \\
f(\pi_{u \leftrightarrow v}) &= f(\pi) + \alpha(r_u - s_u) + \alpha(s_v - r_v) + \alpha(a_{u,v} - a_{v,u}).
\end{aligned}
$$

We start with the case $u <_\pi v$, i.e., $\alpha = 1$. Now $d_{u,v}(\pi) = 1$ holds if and only if C2.1, C3, C5.1, or C6 from Table 1 is fulfilled. We deduce in the following equivalent conditions for each case's occurrence.

With $\alpha = 1$, we have $f(\pi_{u \to v}) - f(\pi) = r_u - s_u$ and $f(\pi_{u \leftrightarrow v}) - f(\pi_{u \leftarrow v}) = r_u - s_u + a_{u,v} - a_{v,u}$.

Case C2.1: $f(\pi) = f(\pi_{u \to v})$ and $f(\pi_{u \leftarrow v}) < f(\pi_{u \leftrightarrow v})$.

We have $f(\pi) = f(\pi_{u \to v})$ if and only if $r_u - s_u = 0$. Further, $f(\pi_{u \leftarrow v}) < f(\pi_{u \leftrightarrow v})$ is equivalent to $r_u - s_u + a_{u,v} - a_{v,u} > 0$, that is, $a_{u,v} - a_{v,u} > s_u - r_u$. Note that $a_{u,v} - a_{v,u} \leq 1$. Altogether, C2.1 is equivalent to $a_{u,v} - a_{v,u} = 1$ and $s_u - r_u = 0$.

Case C3: $f(\pi) > f(\pi_{u \to v})$ and $f(\pi_{u \leftarrow v}) \leq f(\pi_{u \leftrightarrow v})$.

We have $f(\pi) > f(\pi_{u \to v})$ if and only if $r_u - s_u < 0$. Further, $f(\pi_{u \leftarrow v}) \leq f(\pi_{u \leftrightarrow v})$ is equivalent to $r_u - s_u + a_{u,v} - a_{v,u} \geq 0$, that is, $a_{u,v} - a_{v,u} \geq s_u - r_u$. Altogether, C3 is equivalent to $a_{u,v} - a_{v,u} \geq s_u - r_u > 0$, that is, $a_{u,v} - a_{v,u} = 1$ and $s_u - r_u = 1$.

We have $f(\pi_{u\leftarrow v}) - f(\pi) = s_v - r_v$ and $f(\pi_{u\leftrightarrow v}) - f(\pi_{u\rightarrow v}) = s_v - r_v + a_{u,v} - a_{v,u}$.

Case C5.1: $f(\pi) = f(\pi_{u\leftarrow v})$ and $f(\pi_{u\rightarrow v}) < f(\pi_{u\leftrightarrow v})$.

We have $f(\pi) = f(\pi_{u\leftarrow v})$ if and only if $s_v - r_v = 0$. Further, $f(\pi_{u\rightarrow v}) < f(\pi_{u\leftrightarrow v})$ is equivalent to $s_v - r_v + a_{u,v} - a_{v,u} > 0$, that is, $a_{u,v} - a_{v,u} > r_v - s_v$. Altogether, C5.1 is equivalent to $a_{u,v} - a_{v,u} = 1$ and $r_v - s_v = 0$.

Case C6: $f(\pi) > f(\pi_{u\leftarrow v})$ and $f(\pi_{u\rightarrow v}) \leq f(\pi_{u\leftrightarrow v})$.

We have $f(\pi) > f(\pi_{u\leftarrow v})$ if and only if $s_v - r_v < 0$. Further, $f(\pi_{u\rightarrow v}) \leq f(\pi_{u\leftrightarrow v})$ is equivalent to $s_v - r_v + a_{u,v} - a_{v,u} \geq 0$, that is, $a_{u,v} - a_{v,u} \geq r_v - s_v$. Altogether, C6 is equivalent to $a_{u,v} - a_{v,u} \geq r_v - s_v > 0$, that is, $a_{u,v} - a_{v,u} = 1$ and $r_v - s_v = 1$. Equivalently, $s_v - r_v = -1$.

We consider the case $v <_\pi u$. Now $d_{u,v}(\pi) = 1$ holds if and only if C1, C2.2, C4, or C5.2 from Table 1 is fulfilled.

With $\alpha = -1$, we have $f(\pi_{u\rightarrow v}) - f(\pi) = s_u - r_u$ and $f(\pi_{u\leftrightarrow v}) - f(\pi_{u\leftarrow v}) = s_u - r_u + a_{v,u} - a_{u,v}$.

Case C1: $f(\pi) < f(\pi_{u\rightarrow v})$ and $f(\pi_{u\leftarrow v}) \geq f(\pi_{u\leftrightarrow v})$.

We have $f(\pi) < f(\pi_{u\rightarrow v})$ if and only if $s_u - r_u > 0$. Further, $f(\pi_{u\leftarrow v}) \geq f(\pi_{u\leftrightarrow v})$ is equivalent to $s_u - r_u + a_{u,v} - a_{v,u} \leq 0$, that is, $a_{u,v} - a_{v,u} \geq s_u - r_u$. Altogether, C1 is equivalent to $a_{u,v} - a_{v,u} \geq s_u - r_u > 0$, that is, $a_{u,v} - a_{v,u} = 1$ and $s_u - r_u = 1$.

Case C2.2: $f(\pi) = f(\pi_{u\rightarrow v})$ and $f(\pi_{u\leftarrow v}) > f(\pi_{u\leftrightarrow v})$.

We have $f(\pi) = f(\pi_{u\rightarrow v})$ if and only if $s_u - r_u = 0$. Further, $f(\pi_{u\leftarrow v}) > f(\pi_{u\leftrightarrow v})$ is equivalent to $s_u - r_u + a_{v,u} - a_{u,v} < 0$, that is, $a_{u,v} - a_{v,u} > s_u - r_u$. Altogether, C2.2 is equivalent to $a_{u,v} - a_{v,u} = 1$ and $s_u - r_u = 0$.

We have $f(\pi_{u\leftarrow v}) - f(\pi) = r_v - s_v$ and $f(\pi_{u\leftrightarrow v}) - f(\pi_{u\rightarrow v}) = r_v - s_v + a_{v,u} - a_{u,v}$.

Case C4: $f(\pi) < f(\pi_{u\leftarrow v})$ and $f(\pi_{u\rightarrow v}) \geq f(\pi_{u\leftrightarrow v})$.

We have $f(\pi) < f(\pi_{u\leftarrow v})$ if and only if $r_v - s_v > 0$. Further, $f(\pi_{u\rightarrow v}) \geq f(\pi_{u\leftrightarrow v})$ is equivalent to $r_v - s_v + a_{v,u} - a_{u,v} \leq 0$, that is, $a_{u,v} - a_{v,u} \geq r_v - s_v$. Altogether, C4 is equivalent to $a_{u,v} - a_{v,u} \geq r_v - s_v > 0$, that is, $a_{u,v} - a_{v,u} = 1$ and $r_v - s_v = 1$. Equivalently, $s_v - r_v = -1$.

Case C5.2: $f(\pi) = f(\pi_{u\leftarrow v})$ and $f(\pi_{u\rightarrow v}) > f(\pi_{u\leftrightarrow v})$.

We have $f(\pi) = f(\pi_{u\leftarrow v})$ if and only if $r_v - s_v = 0$. Further, $f(\pi_{u\rightarrow v}) > f(\pi_{u\leftrightarrow v})$ is equivalent to $r_v - s_v + a_{v,u} - a_{u,v} < 0$, that is, $a_{u,v} - a_{v,u} > r_v - s_v$. Altogether, C5.2 is equivalent to $a_{u,v} - a_{v,u} = 1$ and $r_v - s_v = 0$.

This finishes the proof. □

Lemma 4.2. *Let $G = (V, E)$ be a DAG with adjacency matrix A and let $D \subseteq A$ be a DSM. Let $u, v \in V$ and $\pi \in \Pi(V)$. If all elements $x \in W_\pi^-(u, v) \cup \{v\}$ that have a DSM-path from or to u with respect to A and π also have such a DSM-path with respect to D and π, then the DSM-jump with main jump from u to v is accepted together with all repair jumps.*

Proof of Lemma 4.2. Assume $u <_\pi v$, the other case is analogous. We start by showing that all repair jumps of the DSM-jump of u, v are accepted. Assume otherwise and let $x \in W_\pi^-(u, v)$ be the first element with an unsuccessful repair jump. The jump of x to v is accepted as long as the objective value does not worsen. It can only do so if there is an element x' with $x <_\pi x' \leq_\pi v$ with an arc $(x, x') \in E$ that is reversed by the jump. As x is part of a repair jump, there must be a DSM-path from u to x. Combined with the arc (x, x') and since $x <_\pi x'$, there is a DSM-path between u and x' in A and π. By assumption, there must therefore be a DSM-path from u to x' also in D. Therefore, x' must have been part of an earlier repair jump, as it is closer to v. As we assumed x to be the first element that was not jumped successfully, x' was jumped successfully. Since j is updated accordingly in Algorithm 2, the repair jump of x cannot move it beyond x', i. e., they preserve their relative order—a contradiction. Thus, all repair jumps are accepted.

Now, the DSM-jump of u to v cannot worsen the fitness: All elements inducing a back arc to u right after the main jump have a DSM-path from u in D and π. Thus, their repair jumps are successful and their relative order to u is preserved. □

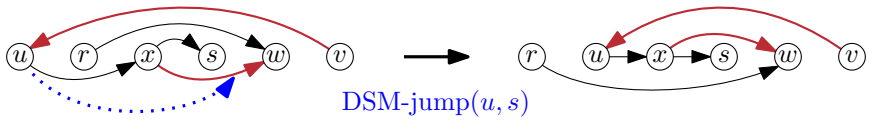


Figure 5: An illustration of the linkage learned in the proof of Theorem 4.3. Known linkages are drawn in black ink and unknown ones in red ink. After the DSM-jump of u right before w is performed, the linkage of x and w is revealed since w has no interaction with the vertices in between.

Theorem 4.3. *DSM-RLS computes a topological ordering of any DAG G with n vertices and m edges in expected $O(n^2 m)$ iterations and $O(n^4 m)$ fitness evaluations.*

Proof of Theorem 4.3. We show that if the algorithm has not found an optimal solution, it has a chance of at least $1/n^2$ to improve the fitness or to detect a new linkage. Consider the current permutation $\pi^{(t)}$ and the current DSM $D^{(t)} = D(\cup_{s \leq t}\{\pi^{(s)}\})$. Let A be the adjacency matrix of G. Let $u <_t v$ be a distance-wise closest pair with $(v, u) \in E$ a back arc. Obviously, such a pair exists if $\pi^{(t)}$ is non-optimal. Let $w \in V$ with $u <_t w <_t v$ be the first element right of u with a DSM-path from u to w in A, which has no DSM-path from u in $D^{(t)}$; cf. Figure 5.

Consider a DSM-jump of u to the position $(\pi^{(t)})_w^{-1} - 1$ one step left of w at time t. This DSM-jump has probability at least $1/n^2$. By Lemma 4.2, this DSM-jump is accepted together with all repair jumps, i. e., the jumps of all elements to which u has a DSM-path in $D^{(t)}$. We show that $\pi^{(t+1)}$ reveals new linkage. Since u jumped over all elements strictly between u and w in $\pi^{(t)}$ it had no DSM-path to, there exists a DSM-path from u to any element in $W^-_{\pi^{(t+1)}}(u, w)$. Further, for some $x \in W^-_{\pi^{(t+1)}}(u, w) \cup \{u\}$ there is an arc $(x, w) \in E$; let x be the rightmost such element. Observe that x lies on a DSM-path from u to w in A as otherwise it would have been moved as part of a repair jump. By the choices of x and w, there is no edge from any vertex $y \in W^-_\pi(x, w)$ strictly between x and w to w. Since there cannot be a back arc from w to any such y as this would be shorter than (u, v), this implies that $\sum_{y \in W^-_\pi(x,w)}(a_{y,w} - a_{w,y}) = 0$. Since $a_{x,w} - a_{w,x} = 1$, the linkage is learned by Lemma 4.1, i. e., $d^{(t+1)}_{x,w} = 1$ and the linkage was unknown in $D^{(t)}$ by assumption.

Once all DSM-paths between u and the elements strictly between u and v have been learned, a DSM-jump from u to v is accepted and correctly orders these elements by Lemma 4.2. Note that this strictly improves the fitness as the back arc from v to u is resolved and due to the acyclicity, no new back arcs can arise. This occurs with probability at least $1/n^2$. Consequently, as long as the optimum has not been reached, in every step there is a probability of at least $1/n^2$ of either discovering a new linkage or improving the objective value. Hence, in expectation, one of these events happens every $O(n^2)$ steps. There are $O(m)$ entries in the ground truth, and the objective value is bounded by $O(m)$, so the time required to reach the optimum is $O(n^2 m)$. Since computing the DSM in each iteration requires $O(n^2)$ function evaluations, the algorithm obtains a topological ordering after $O(n^4 m)$ function evaluations in expectation. □

D Additional data for Section 5.

In this section we report additional results for the plots of Figure 4. In particular, Table 2 to Table 5 show how many of the 100 runs found the optimum, the median runtime, the number of generations, and the number of fitness evaluations. Note that the latter two are almost identical for RLS as in every generation exactly one fitness evaluation is performed. For DSM-RLS, there are additional values showing the fitness evaluations needed for comparing jumps versus the fitness evaluations needed for the DSM as well as the ratio of how many entries of the DSM were revealed in the first iteration of computing the DSM.

There are a few noteworthy observations. The data reveals that the success rate of RLS deteriorates rapidly as the number of subdivisions increases, whereas DSM-RLS maintains consistently high success rates. These results further illustrate the robustness of the proposed approach. Since also cyclic instances can be solved more consistently in very few generations with DSM-RLS, this indicates that the DSM-jump may be useful also for general FAS or linear ordering problems. However,

the instance class and size are rather limited and further tests are necessary to confirm this claim. Further, the number of generations and therefore the number of jumps performed by DSM-RLS is smaller by many orders of magnitude: while RLS performs up to 10^8 generations in 60 seconds, DSM-RLS can only perform a few thousand and this overhead comes almost only from computing the DSM. Since almost the complete DSM is found in the first generation it seems safe to assume that the majority of evaluations is unnecessary, thus leaving a large space for algorithmic improvement. In particular, in a grey-box setting, where the matrix is known before, the immense computational overhead of computing the DSM entries can be avoided entirely.

Table 2: RLS – trees (100 runs per configuration).

k	Success rate	Median runtime (s)	Mean generations	Mean fitness evals
0	1.00	0.003	7.77×10^3	7.78×10^3
1	0.98	0.052	1.32×10^7	1.32×10^7
2	0.70	1.94	1.05×10^8	1.05×10^8
3	0.40	60.00	1.72×10^8	1.72×10^8
4	0.18	60.00	1.98×10^8	1.98×10^8
5	0.02	60.00	2.04×10^8	2.04×10^8

Table 3: DSM-RLS – trees, with evaluation-phase breakdown (100 runs per configuration).

k	Success rate	Median runtime (s)	Mean generations	Mean fitness evals	Mean jump evals	Jump ratio	Initial fill ratio
0	1.00	0.065	1.55×10^2	2.36×10^5	3.80×10^2	0.0016	0.956
1	1.00	0.505	4.96×10^2	3.02×10^6	1.79×10^3	0.0006	1.000
2	1.00	2.48	9.35×10^2	1.28×10^7	4.42×10^3	0.0003	1.000
3	1.00	8.26	1.43×10^3	3.49×10^7	7.94×10^3	0.0002	1.000
4	0.99	21.67	1.99×10^3	7.56×10^7	1.31×10^4	0.0002	1.000
5	0.86	44.73	2.24×10^3	1.23×10^8	1.66×10^4	0.0001	1.000

Table 4: RLS – uniform graphs (100 runs per configuration).

k	Success rate	Median runtime (s)	Mean generations	Mean fitness evals
0	1.00	0.002	1.51×10^3	1.51×10^3
1	1.00	0.090	1.31×10^7	1.31×10^7
2	0.86	24.60	1.89×10^9	1.89×10^9
3	0.37	1800.00	5.63×10^9	5.63×10^9
4	0.08	1800.00	6.18×10^9	6.18×10^9
5	0.02	1800.00	5.80×10^9	5.80×10^9

Table 5: DSM-RLS – uniform graphs, with evaluation-phase breakdown (100 runs per configuration).

k	Success rate	Median runtime (s)	Mean generations	Mean fitness evals	Mean jump evals	Jump ratio	Initial fill ratio
0	1.00	0.018	1.81×10^2	6.16×10^4	9.84×10^2	0.0150	0.849
1	1.00	0.418	1.24×10^3	5.69×10^6	1.34×10^4	0.0021	1.000
2	1.00	3.96	6.30×10^3	3.15×10^7	1.04×10^5	0.0011	1.000
3	0.99	18.07	1.10×10^4	1.50×10^8	2.35×10^5	0.0007	1.000
4	0.95	71.83	1.65×10^4	2.58×10^8	4.35×10^5	0.0005	1.000
5	0.89	180.52	1.58×10^4	3.06×10^8	5.08×10^5	0.0004	1.000