Sorting With Forbidden Intermediates

Carlo Comin, Anthony Labarre, Romeo Rizzi, Stéphane Vialette

Sorting With Forbidden Intermediates

Carlo Comin* Anthony Labarre† Romeo Rizzi‡
Stéphane Vialette§

Abstract

A wide range of applications, most notably in comparative genomics, involve the computation of a shortest sorting sequence of operations for a given permutation, where the set of allowed operations is fixed beforehand. Such sequences are useful for instance when reconstructing potential scenarios of evolution between species, or when trying to assess their similarity. We revisit those problems by adding a new constraint on the sequences to be computed: they must *avoid* a given set of *forbidden intermediates*, which correspond to species that cannot exist because the mutations that would be involved in their creation are lethal. We initiate this study by focusing on the case where the only mutations that can occur are exchanges of any two elements in the permutations, and give a polynomial time algorithm for solving that problem when the permutation to sort is an involution.

1 Introduction

Computing distances between permutations, or sequences of operations that transform them into one another, are two generic problems that arise in a wide range of applications, including comparative genomics [6], ranking [4], and interconnection network design [15]. Those problems are well-known to reduce to constrained sorting problems of the following form: given a permutation π and a set S of allowed operations, find a sequence of elements from S that sorts π and is as short as possible. In the context of comparative genomics, the sequence to be reconstructed yields a possible scenario of evolution between the genomes represented by π and the target identity permutation ι, where all permutations obtained inbetween are successive descendants of π (and ancestors of ι). The many possible choices that exist for S, as well as other constraints or cost functions with which they can be combined, have given rise to a tremendous number of variants whose algorithmic and mathematical aspects have now been studied for decades [6]. Specific issues that biologists feel need to be addressed to improve the applicability of these results in a biological context include: 1) the oversimplicity of the model (permutations do

*Department of Mathematics, University of Trento, Italy. E-mail: Carlo.Comin@unitn.it
†Université Paris-Est, LIGM (UMR 8049), UPEM, CNRS, ESIEE, ENPC, F-77454, Marne-la-Vallée, France. E-mail: Anthony.Labarre@u-pem.fr
‡Department of Computer Science, University of Verona, Italy. Romeo.Rizzi@univr.it
§Université Paris-Est, LIGM (UMR 8049), UPEM, CNRS, ESIEE, ENPC, F-77454, Marne-la-Vallée, France. E-mail: Stephane.Vialette@u-pem.fr

not take duplications into account), 2) the rigid definition of allowed operations, which fails to capture the complexity of evolution, and 3) the complexity of the resulting problems, where algorithmic hardness results abound even for deceivingly simple problems. A large body of work has been devoted to addressing those issues, namely by proposing richer models for genomes, encompassing several operations with different weights [6]. Some approaches for increasing the reliability of rearrangement methods by adding additional biologically motivated constraints have been investigated (see for example [2] for conserved intervals, [7] for restricting the set of allowed inversions and [1] for preserving the number of inversions in the scenario which commute with all common intervals). However, another critical issue has apparently been overlooked: to the best of our knowledge, no model takes into account the fact that the solutions it produces may involve allele mutations that are lethal to the organism on which they act. *Lethals* are usually a result of mutations in genes that are essential to growth or development [9]; they have been known to occur for more than a century [3], as they were first discovered by Cuénot in 1905 while studying the inheritance of coat colour in mice. As a consequence, solutions that may be perfectly valid from a mathematical point of view should nonetheless be rejected on the grounds that some of the intermediate ancestors they produce are nonviable and can therefore not have had any descendants. We revisit the family of problems mentioned above by adding a natural constraint which, as far as we know, has not been previously considered in this form (see e.g. [2, 7, 1] for connected attempts): namely, the presence of a set of forbidden intermediate permutations, which the sorting sequence that we seek must avoid. We refer to this family of problems as GUIDED SORTING problems, since they take additional guidance into account. In this paper we focus our study on the case where only *exchanges* (i.e., algebraic transpositions) are allowed; furthermore, we simplify the problem by demanding that the solutions we seek be *optimal* in the sense that no shorter sorting sequence of exchanges exists even when no intermediate permutation is forbidden. We choose to focus on exchanges because of their connection to the underlying *disjoint cycle structure* of permutations, which plays an important role in many related sorting problems where a similar cycle-based approach, using this time the ubiquitous *breakpoint graph*, has proved extremely fruitful [14]. Therefore, we believe that progress on this particular variant will be helpful when attempting to solve related variants based on more complex operations. Our main contribution in this work is a polynomial time algorithm for solving GUIDED SORTING by exchanges when the permutation to sort is an *involution*. We show that, in that specific case, the space of all feasible sorting sequences admits a suitable description in terms of directed (s, t)-paths in hypercube graphs. We achieve this result by reducing GUIDED SORTING to the problem of finding directed (s, t)-paths that avoid a prescribed set $\mathcal{F} \subseteq V$ of *forbidden vertices*. Our main contribution, therefore, consists in solving this latter problem in time polynomial in just the encoding of $\mathcal{F}$ if G is constrained to be a *hypercube* graph, which is a novel algorithmic result that may be of independent interest. Specific properties that will be described later on [10, 16] allow us to avoid the full construction of that graph, which would lead to an exponential time algorithm. We should mention that constrained variants of the (s, t)-connectivity problem have been studied already to some extent. For instance, already in the '70s, motivated by some problems in the field of automatic software testing and validation, Krause et al. [13] introduced the *path avoiding forbidden pairs* problem,

namely, that of finding a directed (s,t)-path in a graph $G = (V, E)$ that contains at most one vertex from each pair in a prescribed set $\mathcal{F} \subseteq V \times V$ of *forbidden pairs* of vertices. Gabow et al. [8] proved that the problem is NP-complete on DAGs. A number of special cases were shown to admit polynomial time algorithms, e.g. Yinnone [18] studied the problem in directed graphs under a *skew-symmetry* condition. However, the involved techniques and the related results do not extend to our problem, for which we are aware of no previously known algorithm that runs in time polynomial in just the encoding of $\mathcal{F}$.

2 Background and Notation

Our aim is to sort a given permutation π using a predefined set of allowed operations, specified as a generating set S of the symmetric group $\mathfrak{S}_n$. We seek a sorting sequence that uses only elements from S and: 1) *avoids* a given set $\mathcal{F}$ of *forbidden permutations*, i.e. no intermediary permutation produced by applying the operations specified by the sorting sequence belongs to $\mathcal{F}$, and 2) is *optimal*, i.e. no shorter sorting sequence exists for π even if $\mathcal{F} = \emptyset$. We refer to the general problem of finding a sorting sequence under these constraints as GUIDED SORTING, and restrict in this paper the allowed operations to *exchanges* of any two elements (i.e. *algebraic transpositions*). For instance, let $\pi = \langle 2\ 3\ 1\ 4\rangle$ and $\mathcal{F} = \{\langle 1\ 3\ 2\ 4\rangle, \langle 3\ 2\ 1\ 4\rangle\}$. Then $\langle 2\ 3\ 1\ 4\rangle \mapsto \langle 2\ 1\ 3\ 4\rangle \mapsto \langle 1\ 2\ 3\ 4\rangle$ is a valid solution since it is optimal and avoids $\mathcal{F}$, but neither $\langle 2\ 3\ 1\ 4\rangle \mapsto \langle 4\ 3\ 1\ 2\rangle \mapsto \langle 4\ 3\ 2\ 1\rangle \mapsto \langle 4\ 2\ 3\ 1\rangle \mapsto \langle 1\ 2\ 3\ 4\rangle$ nor $\langle 2\ 3\ 1\ 4\rangle \mapsto \langle 1\ 3\ 2\ 4\rangle \mapsto \langle 1\ 2\ 3\ 4\rangle$ can be accepted: the former is too long, and the latter does not avoid $\mathcal{F}$.

We use standard notions and notation from graph theory (see e.g. [5] for undefined concepts), using $\{u, v\}$ (resp. (u, v)) to denote the edge (resp. arc) between vertices u and v of an undirected (resp. directed) graph $G = (V, E)$. All graphs we consider are *simple*: they contain neither loops nor parallel edges. If $\mathcal{F} \subseteq V$, a directed path $\mathsf{p} = v_0 v_1 \cdots v_n$ *avoids* $\mathcal{F}$ when $v_i \notin \mathcal{F}$ for every i. If $\mathcal{S} \subseteq V$ and $\mathcal{T} \subseteq V$, we say that a directed path p *goes from* $\mathcal{S}$ *to* $\mathcal{T}$ *in* G when p starts from some s in $\mathcal{S}$ and ends at some t in $\mathcal{T}$. When G is directed, we partition the neighbourhood $N(u)$ of a vertex u into the sets $N^{\mathtt{out}}(u) = \{v \in V \mid (u, v) \in E\}$ and $N^{\mathtt{in}}(u) = \{v \in V \mid (v, u) \in E\}$. Some of our graphs may be vertex-labelled, using any injective mapping $\ell : V \to \mathbb{N}$. For any $n \in \mathbb{N}$, $\wp_n = \wp([n])$ denotes the power set of $[n]$. The *hypercube graph on ground set* $[n]$, denoted by $\mathcal{H}_n$, is the graph with vertex set $\wp_n$ and in which the arc (U, V) connects vertices $U, V \subseteq [n]$ if there exists some $q \in [n]$ such that $U = V \setminus \{q\}$. If $S, T \in \wp_n$ and $|S| \leq |T|$, then $d_{S,T} = |T| - |S|$ is the *distance* between S and T. Finally, $\mathcal{H}_n^{(i)}$ denotes the family of all subsets of $\wp_n$ of size i.

3 Solving GUIDED SORTING For Involutions

The *Cayley graph* $\Gamma(\mathfrak{S}_n, S)$ of $\mathfrak{S}_n$ for a given generating set S of $\mathfrak{S}_n$ contains a vertex for each permutation in $\mathfrak{S}_n$ and an edge between any two permutations that can be obtained from one another using one element from S. A naïve approach for solving any variant of the GUIDED SORTING problem would build the part of $\Gamma(\mathfrak{S}_n, S)$ that is needed (i.e. without the elements of $\mathcal{F}$), then run a shortest path

algorithm to compute an optimal sequence that avoids all elements of $\mathcal{F}$. This is highly impractical, since the size of Γ is exponential in n.

We describe in this section a polynomial time algorithm in the case of exchanges if π is an *involution*, i.e. a permutation such that for each $1 \leq i \leq n$, either $\pi_i = i$ or there exists an index j such that $\pi_i = j$ and $\pi_j = i$. From our point of view, involutions reduce to collections of disjoint pairs of elements that each need to be swapped by an exchange until we obtain the identity permutation, and the only forbidden permutations that could be produced by an optimal sorting sequence are involutions whose pairs of unsorted elements all appear in π. Therefore, we can reformulate our GUIDED SORTING problem in that setting as that of finding a directed (π, ι)-path in $\mathcal{H}_n$ that avoids all vertices in $\mathcal{F}$, where the permutation to sort π corresponds to the bottom vertex $\emptyset$ of $\mathcal{H}_n$ and the identity permutation ι corresponds to the top vertex $[n]$ of $\mathcal{H}_n$. We shall focus on the following problem from here on.

PROBLEM: HY-STCON.

INPUT: the size $n \in \mathbb{N}$ of the underlying ground set $[n]$, a family of *forbidden vertices* $\mathcal{F} \subseteq \wp_n$, a *source* set $S \in \wp_n$ and a *target* set $T \in \wp_n$.
DECISION-TASK: Decide whether there exists a directed path p in $\mathcal{H}_n$ that goes from source S to target T avoiding $\mathcal{F}$;
SEARCH-TASK: Compute a directed path p in $\mathcal{H}_n$ that goes from source S to target T avoiding $\mathcal{F}$, provided that at least one such path exists.

We will show how to solve HY-STCON in time polynomial in $|\mathcal{F}|$ and n. The algorithm mainly consists in the continuous iteration of two phases:

1. *Double-BFS.* This phase explores the outgoing neighbourhood of the source S by a breadth-first search denoted by $\mathtt{BFS}_\uparrow$ going from lower to higher levels of $\mathcal{H}_n$ while avoiding the vertices in $\mathcal{F}$. $\mathtt{BFS}_\uparrow$ collects a certain (polynomially bounded) amount of visited vertices. Symmetrically, the incoming neighbourhood of the target vertex T is also explored by another breadth-first search $\mathtt{BFS}_\downarrow$ going from higher to lower levels of $\mathcal{H}_n$ while avoiding the vertices in $\mathcal{F}$, also collecting a certain (polynomially bounded) amount of visited vertices.

2. *Compression.* If a valid solution has not yet been determined, then a compression technique is devised in order to shrink the size of the remaining search space. This is possible thanks to some nice regularities of the search space and to certain connectivity properties of hypercube graphs [10, 16]. This allows us to reduce the search space in a suitable way and, therefore, to continue with the Double-BFS phase in order to keep the search towards valid solutions going.

Our main contribution is summarized in the following theorem. We devote the rest of this section to an in-depth description of the algorithms it mentions[1].

Theorem 1. *Concerning the* HY-STCON *problem, the following propositions hold on any input $\langle S, T, \mathcal{F}, n\rangle$, where $d_{S,T}$ is the distance between S and T.*

1. *There exists an algorithm for solving the* DECISION-TASK *of* HY-STCON *within* $O(\min(\sqrt{|\mathcal{F}|\, d_{S,T}}\, n, |\mathcal{F}|)\, |\mathcal{F}|^2\, d_{S,T}^4\, n^2)$ *time.*

[1]Dear reviewers: see Appendix B.2 for correctness and Appendix B.3 for complexity.

2. *There exists an algorithm for solving the* Search-Task *of* Hy-stCon *within* $O(\min(\sqrt{|\mathcal{F}|\, d_{S,T}\, n}, |\mathcal{F}|)\, |\mathcal{F}|^2\, d_{S,T}^4\, n^2 + |\mathcal{F}|^{5/2} n^{3/2} d_{S,T})$ *time.*

3.1 On Vertex-Disjoint Paths in Hypercube Graphs

The proof of Theorem 1 relies on connectivity properties of hypercube graphs [10]. The next result, which proves the existence of a family of certain vertex-disjoint paths in $\mathcal{H}_n$ that are called *Lehman-Ron paths*, will be particularly useful.

Theorem 2 (Lehman, Ron [16])**.** *Given $n, m \in \mathbb{N}$, let $\mathcal{R} \subseteq \mathcal{H}_n^{(r)}$ and $\mathcal{S} \subseteq \mathcal{H}_n^{(s)}$ with $|\mathcal{R}| = |\mathcal{S}| = m$ and $0 \leq r < s \leq n$. Assume there exists a bijection $\varphi : \mathcal{S} \to \mathcal{R}$ such that $\varphi(S) \subset S$ for every $S \in \mathcal{S}$. Then there exist m vertex-disjoint directed paths in $\mathcal{H}_n$ whose union contains all the subsets in $\mathcal{S}$ and $\mathcal{R}$.*

We call tuples $\langle \mathcal{R}, \mathcal{S}, \varphi, n \rangle$ that satisfy the hypotheses of Theorem 2 *Lehman-Ron* tuples, and we refer to the quantity $d = s - r$ as the *distance* between $\mathcal{R} \subseteq \mathcal{H}_n^{(r)}$ and $\mathcal{S} \subseteq \mathcal{H}_n^{(s)}$. Lehman and Ron [16] give an elementary inductive proof of Theorem 2. A careful and in-depth analysis of their proof, from the algorithmic perspective, yields a polynomial time algorithm for computing all the Lehman-Ron paths.

Theorem 3. *There exists an algorithm for computing all Lehman-Ron paths within time $O\big(m^{5/2} n^{3/2} d\big)$ on any Lehman-Ron input $\langle \mathcal{R}, \mathcal{S}, \varphi, n \rangle$ with $|\mathcal{R}| = |\mathcal{S}| = m$, where d is the distance between $\mathcal{R}$ and $\mathcal{S}$ and n is the size of the underlying ground set.*

In an extended version[2] we provide all the details of the above mentioned algorithm as well as a proof of the time complexity stated in Theorem 3, in which Menger's vertex-connectivity theorem [5] and Hopcroft-Karp's algorithm [11] for maximum cardinality matching in *undirected* bipartite graphs play a major role.

3.2 A Polynomial Time Algorithm For Solving Hy-stCon

We now describe a polynomial time algorithm for solving Hy-stCon, called `solve_Hy-stCon()`, which takes as input an instance $\langle S, T, \mathcal{F}, n \rangle$ of Hy-stCon, and returns a pair $\langle \texttt{YES}, \texttt{p} \rangle$ where `p` is a directed path in $\mathcal{H}_n$ that goes from source S to target T avoiding $\mathcal{F}$ if such a path exists (otherwise, the algorithm simply returns `NO`). Algorithm 1 shows the pseudocode for that procedure. The rationale at the base of `solve_Hy-stCon()` consists in the continuous iteration of two major phases: `double-bfs_phase()` (line 5) and `compression_ phase()` (line 11). Throughout computation, both phases alternate repeatedly until a final state of termination is eventually reached (either at line 7, line 10 or line 12). At that point, the algorithm either returns a pair $\langle \texttt{YES}, \texttt{p} \rangle$ where `p` is the sought directed path, or a negative response `NO` instead[3]. We now describe both phases in more detail, and give the corresponding pseudocode.

[2] See Appendix B.1 for all the technical details.

[3] See Appendix B.2-B.3 for details on correctness and complexity.

Algorithm 1: solving the Hy-stCon problem.

Procedure *solve*_Hy-stCon$(S, T, \mathcal{F}, n)$
  Input: an instance $\langle S, T, \mathcal{F}, n\rangle$ of Hy-stCon.
  Output: a pair $\langle$`YES`, p$\rangle$ where the path p is a solution to Hy-stCon if such a path exists, `NO` otherwise.
1 $d_{S,T} \leftarrow |T| - |S|$; // let $d_{S,T}$ be the distance between S and T
2 $\mathcal{S} \leftarrow \{S\}$; $\ell_\uparrow \leftarrow 0$; // initialize the *frontier* $\mathcal{S}$ and its *level counter* $\ell_\uparrow$
3 $\mathcal{T} \leftarrow \{T\}$; $\ell_\downarrow \leftarrow 0$; // initialize the *frontier* $\mathcal{T}$ and its *level counter* $\ell_\downarrow$
4 **while** *TRUE* **do**
5 $\langle \mathcal{S}, \mathcal{T}, \ell_\uparrow, \ell_\downarrow\rangle \leftarrow$ `double-bfs_phase`$(\mathcal{S}, \mathcal{T}, \mathcal{F}, \ell_\uparrow, \ell_\downarrow, d_{S,T}, n)$;
6 **if** $\mathcal{S} = \emptyset$ *OR* $\mathcal{T} = \emptyset$ *OR* $(\ell_\uparrow + \ell_\downarrow = d_{S,T}$ *AND* $\mathcal{S} \cap \mathcal{T} = \emptyset)$ **then**
7 **return** *NO*;
8 **if** $\ell_\uparrow + \ell_\downarrow = d_{S,T}$ *AND* $\mathcal{S} \cap \mathcal{T} \neq \emptyset$ **then**
9 p $\leftarrow$ `reconstruct_path`$(\mathcal{S}, \mathcal{T}, n)$;
10 **return** $\langle$*YES*, p$\rangle$;
11 `returned_val` $\leftarrow$ `compression_phase`$(\mathcal{S}, \mathcal{T}, \mathcal{F}, \ell_\uparrow, \ell_\downarrow, d_{S,T}, n)$;
12 **if** *returned_val* $= \langle$*YES*, p$\rangle$ **then return** p;
13 **else** $\mathcal{T} \leftarrow$ `returned_val`;

Algorithm 2: Breadth-First-Search phases of `solve_Hy-stCon()`.

Procedure *double-bfs_phase*$(\mathcal{S}, \mathcal{T}, \mathcal{F}, \ell_\uparrow, \ell_\downarrow, d_{S,T}, n)$
1 $\langle \mathcal{S}^*, \ell^*_\uparrow\rangle \leftarrow$ `bfs_phase`$(\mathcal{S}, \mathcal{F}, \ell_\uparrow, \ell_\downarrow,$ `out`$, d_{S,T}, n)$; // BFS$_\uparrow$ phase
2 $\langle \mathcal{T}^*, \ell^*_\downarrow\rangle \leftarrow$ `bfs_phase`$(\mathcal{T}, \mathcal{F}, \ell_\downarrow, \ell^*_\uparrow,$ `in`$, d_{S,T}, n)$; // BFS$_\downarrow$ phase
3 **return** $\langle \mathcal{S}^*, \mathcal{T}^*, \ell^*_\uparrow, \ell^*_\downarrow\rangle$;

SubProcedure *bfs_phase*$(\mathcal{X}, \mathcal{F}, \ell_x, \ell_y,$ *drt*$, d_{S,T}, n)$
1 **while** $1 \leq |\mathcal{X}| \leq |\mathcal{F}|\, d_{S,T}$ *AND* $\ell_x + \ell_y < d_{S,T}$ **do**
2 $\mathcal{X} \leftarrow$ `next_step_bfs`$(\mathcal{X}, \mathcal{F},$ `drt`$, n)$;
3 $\ell_x \leftarrow \ell_x + 1$;
4 **return** $\langle \mathcal{X}, \ell_x\rangle$;

SubProcedure *next_step_bfs*$(\mathcal{X}, \mathcal{F},$ *drt*$, n)$
1 $\mathcal{X}' \leftarrow \emptyset$;
2 **foreach** $v \in \mathcal{X}$ **do**
3 $\mathcal{X}' \leftarrow \mathcal{X}' \cup N^{\texttt{drt}}(v) \setminus \mathcal{F}$; // $N^{\texttt{drt}}$ is $N^{\texttt{in}}$ if `drt` $=$ `in`, otherwise it is $N^{\texttt{out}}$
4 **return** $\mathcal{X}'$;

Breadth-First Search phases. The first search BFS$_\uparrow$ starts from the source vertex S and moves upward, from lower to higher levels of $\mathcal{H}_n$. Meanwhile, it collects a certain (polynomially bounded) amount of vertices that do not lie in $\mathcal{F}$. In particular, at the end of any BFS$_\uparrow$ phase, the number of collected vertices will always lie between $|\mathcal{F}|\, d_{S,T} + 1$ and $|\mathcal{F}|\, d_{S,T}\, n$ (see line 1 of `bfs_phase()`). The set $\mathcal{S}$ of vertices collected at the end of BFS$_\uparrow$ is called the *(source) frontier* of BFS$_\uparrow$. All vertices within $\mathcal{S}$ have the same cardinality, i.e. $|X_1| = |X_2|$ for every $X_1, X_2 \in \mathcal{S}$. Also, the procedure keeps track of the highest level of depth $\ell_\uparrow$ that is reached during BFS$_\uparrow$. Thus, $\ell_\uparrow$ corresponds to the distance between the source vertex S and the current frontier $\mathcal{S}$, formally, $\ell_\uparrow = |X| - |S|$ for every $X \in \mathcal{S}$. Since at the beginning of the computation BFS$_\uparrow$ starts from the source vertex S, `solve_Hy-stCon()` initializes $\mathcal{S}$ to $\{S\}$ and $\ell_\uparrow$ to 0 at line 2.

Similarly, the second search BFS$_\downarrow$ starts from the target vertex T and moves

downward, from higher to lower levels of $\mathcal{H}_n$, also collecting a certain (polynomially bounded) amount of vertices that do not lie in $\mathcal{F}$. As in the previous case, this amount will always lie between $|\mathcal{F}|\, d_{S,T} + 1$ and $|\mathcal{F}|\, d_{S,T}\, n$. The set $\mathcal{T}$ of vertices collected at the end of $\texttt{BFS}_\downarrow$ is called the *(target) frontier* of $\texttt{BFS}_\downarrow$. All vertices within $\mathcal{T}$ have the same cardinality. Also, the procedure keeps track of the *lowest* level of depth $\ell_\downarrow$ that $\texttt{BFS}_\downarrow$ has reached. Thus, $\ell_\downarrow$ corresponds to the *distance* between the target vertex T and the frontier $\mathcal{T}$, so that $\ell_\downarrow = |T| - |X|$ for every $X \in \mathcal{T}$. Since at the beginning of the computation, $\texttt{BFS}_\downarrow$ starts from the target vertex T, `solve_`Hy-stCon`()` initializes $\mathcal{T} = \{T\}$ and $\ell_\downarrow = 0$ at line 3. Fig. 1 provides an illustration of `double-bfs_phase()`.

In summary, after any round of `double-bfs_phase()`, we are left with two (possibly empty) frontier sets $\mathcal{S}$ and $\mathcal{T}$. In Algorithm 1, whenever $\mathcal{S} = \emptyset$ or $\mathcal{T} = \emptyset$ holds at line 6, then at least one frontier set could not proceed one level further in $\mathcal{H}_n$ while avoiding $\mathcal{F}$, and thus the procedure halts by returning `NO` at line 7. Similarly, whenever $\ell_\uparrow + \ell_\downarrow = d_{S,T}$ and $\mathcal{S} \cap \mathcal{T} = \emptyset$ holds at line 6, the computation halts by returning `NO` at line 7 — the underlying intuition being that $\mathcal{S}$ and $\mathcal{T}$ have finally reached one another's level of depth without intersecting each other, which means that $\mathcal{H}_n$ contains no directed path from S to T that avoids $\mathcal{F}$[4].

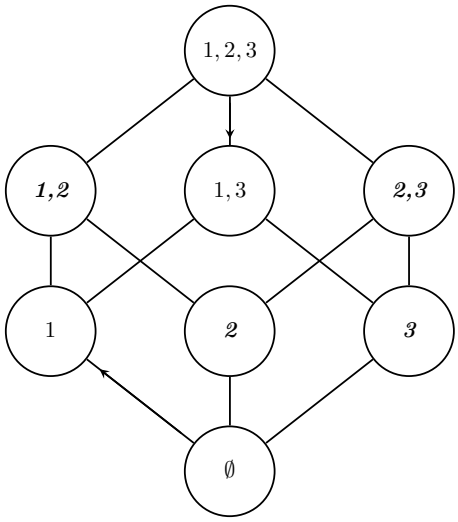

Figure 1: A `double_bfs_phase()` on $\mathcal{H}_3$ that starts from $S = \emptyset$ and $T = \{1, 2, 3\}$. The forbidden vertices are $\mathcal{F} = \{\{2\}, \{3\}, \{1, 2\}, \{2, 3\}\}$, while the edges explored by $\texttt{BFS}_\uparrow$ and $\texttt{BFS}_\downarrow$ are $(\emptyset, \{1\})$ and $(\{1, 2, 3\}, \{1, 3\})$ (respectively).

On the other hand, if both $\ell_\uparrow + \ell_\downarrow = d_{S,T}$ and $\mathcal{S} \cap \mathcal{T} \neq \emptyset$ hold at line 8, then we can prove that for every $S' \in \mathcal{S}$, there exists at least one directed path in $\mathcal{H}_n$ that goes from the source S to S' avoiding $\mathcal{F}$. Similarly, for every $T' \in \mathcal{T}$, there exists at least one directed path in $\mathcal{H}_n$ that goes from T' to target T avoiding $\mathcal{F}$. Therefore, whenever $\mathcal{S} \cap \mathcal{T} \neq \emptyset$, the algorithm is in the right position to reconstruct a directed path $\texttt{p}$ in $\mathcal{H}_n$ that goes from source S to $\mathcal{S} \cap \mathcal{T}$ and from $\mathcal{S} \cap \mathcal{T}$ to target T avoiding

[4]See Appendix B.2

Algorithm 3: Compression phase of `solve_Hy-stCon()`.

```
   Procedure compression_phase(S, T, F, ℓ↑, ℓ↓, d_{S,T}, n)
1      T' ← ∅;
2      while TRUE do
3          G ← construct_bipartite_graph(S, T, n);
4          M ← compute_max_matching(G, |F| + 1);
5          if |M| > |F| then
6              M_S ← {X ∈ S | ∃ Y ∈ T s.t (X, Y) ∈ M};
7              M_T ← {Y ∈ T | ∃ X ∈ S s.t. (X, Y) ∈ M};
8              {p_1, ..., p_{|M|}} ← compute_Lehman-Ron_paths(M_S, M_T, M, n);
9              p ← reconstruct_path(S, T, {p_i}_{i=1}^{|M|}, n);
10             return ⟨YES, p⟩;
11         X ← compute_min_vertex_cover(G, M);
12         X_S ← X ∩ S; X_T ← X ∩ T;
13         T' ← T' ∪ X_T;
14         ⟨S, T, ℓ↑, ℓ↓⟩ ← double-bfs_phase(X_S, T, F, ℓ↑, ℓ↓, d_{S,T}, n);
15         if S = ∅ OR (ℓ↓ + ℓ↑ = d_{S,T} AND S ∩ T = ∅ ) then
16             return T';
17         if ℓ↑ + ℓ↓ = d_{S,T} AND S ∩ T ≠ ∅ then
18             p ← reconstruct_path(S, T, n);
19             return ⟨YES, p⟩;
```

$\mathcal{F}$ (line 9). In practice, the reconstruction can be implemented by maintaining a `map` throughout the computation, which associates to every vertex v (possibly visited during the BFSs) the *parent vertex*, `parent`(v), which led to discover v first. As soon as `p` gets constructed, `solve_Hy-stCon()` returns $\langle$`YES`, `p`$\rangle$ at line 10, and the computation halts.

Compression Phase. After `double-bfs_phase()` has completed, the procedure `solve_Hy-stCon()` also needs to handle the case where $\mathcal{S}, \mathcal{T} \neq \emptyset$ and $\ell_\downarrow + \ell_\uparrow < d_{S,T}$. The phase that starts at that point is named `compression_phase()` (see Algorithm 3). This procedure takes as input a tuple $\langle \mathcal{S}, \mathcal{T}, \mathcal{F}, \ell_\uparrow, \ell_\downarrow, d_{S,T}, n \rangle$, where $\mathcal{S}$ and $\mathcal{T}$ are the current frontier sets. Recall that $|\mathcal{T}| > |\mathcal{F}|\, d_{S,T}$ holds due to line 1 of `bfs_phase()`. Also, $\mathcal{F} \subseteq \wp_n$ is the set of forbidden vertices; $\ell_\uparrow$ is the level counter of $\mathcal{S}$ and $\ell_\downarrow$ is that of $\mathcal{T}$; finally $d_{S,T}$ is the distance between the source S and the target T, and n is the size of the ground set. The output returned by `compression_phase()` is either a path `p` that goes from the source S to the target T avoiding $\mathcal{F}$ `or` a subset $\mathcal{T}' \subset \mathcal{T}$ such that the following two basic properties hold: (1) $|\mathcal{T}'| \leq |\mathcal{F}|\, d_{S,T}$, and (2) if `p` is any directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}$ to $\mathcal{T}$ avoiding $\mathcal{F}$, then `p` goes from $\mathcal{S}$ to $\mathcal{T}'$.

This frontier set $\mathcal{T}'$ is dubbed the *compression* of $\mathcal{T}$. The underlying rationale goes as follows. On one hand, because of (1), it is possible to keep the search going on by applying yet another round of `double-bfs_phase()` on input $\mathcal{S}$ and $\mathcal{T}'$ (in fact, the size of $\mathcal{T}$ has been compressed down to $|\mathcal{T}'| \leq |\mathcal{F}|\, d_{S,T}$, thus matching the threshold condition "$|\mathcal{X}| \leq |\mathcal{F}|\, d_{S,T}$" checked at line 1 of `bfs_phase()`). On the other hand, because of (2), it is indeed sufficient to seek for a directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}$ to $\mathcal{T}'$ avoiding $\mathcal{F}$, namely, the search can actually forget about $\mathcal{T} \setminus \mathcal{T}'$ because it leads to a dead end. We now describe `compression_phase()` in more

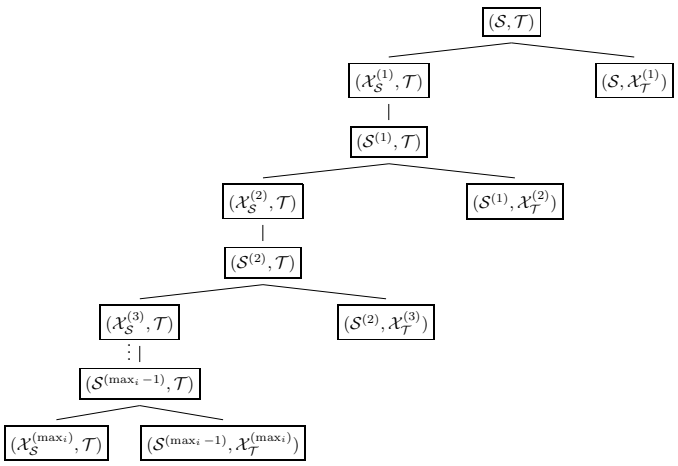

Figure 2: The frontier sets considered during the `compression_phase()`.

details, and give a graphical summary in Fig. 2. The procedure repeatedly builds an undirected bipartite graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}})$, where $V_{\mathcal{G}} = \mathcal{S} \cup \mathcal{T}$ and every vertex $U \in \mathcal{S}$ is adjacent to a vertex $V \in \mathcal{T}$ if and only if $U \subset V$. It then uses the procedure `compute_max_matching()` to find a matching $\mathcal{M}$ of size $|\mathcal{M}| = \min(m^*, |\mathcal{F}| + 1)$, where m^* denotes the size of a maximum cardinality matching of $\mathcal{G}$. In practice, this step can be implemented in the same manner as a maximum cardinality matching procedure, e.g. as Hopcroft-Karp's algorithm [11], although with the following basic variation: if the size of the augmenting matching $\mathcal{M}$ eventually reaches the cut-off value $|\mathcal{F}| + 1$, then `compute_max_matching()` returns $\mathcal{M}$ and halts (i.e. even if $m^* > |\mathcal{F}| + 1$). The next course of action depends on $|\mathcal{M}|$:

1. If $|\mathcal{M}| = |\mathcal{F}| + 1$, then the procedure relies on Theorem 3 to compute a family $\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_{|\mathcal{M}|}$ of $|\mathcal{M}|$ vertex-disjoint directed paths in $\mathcal{H}_n$ that go from $\mathcal{S}$ to $\mathcal{T}$. In order to do that, the procedure considers the subset $\mathcal{M}_{\mathcal{S}} \subseteq \mathcal{S}$ (resp. $\mathcal{M}_{\mathcal{T}} \subseteq \mathcal{T}$) of all vertices in $\mathcal{S}$ (resp. in $\mathcal{T}$) that are incident to some edge in $\mathcal{M}$ (lines 6 and 7). Notice that the matching $\mathcal{M}$ can be viewed as a bijection between $\mathcal{M}_{\mathcal{S}}$ and $\mathcal{M}_{\mathcal{T}}$. Then, the algorithm underlying Theorem 3 gets invoked on input $\langle \mathcal{M}_{\mathcal{S}}, \mathcal{M}_{\mathcal{T}}, \mathcal{M}, n \rangle$ (line 8). Once all the Lehman-Ron paths $\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_{|\mathcal{M}|}$ have been found, it is then possible to reconstruct the sought directed path p in $\mathcal{H}_n$ that goes from source S to target T avoiding $\mathcal{F}$ (line 9). In fact, since $|\mathcal{M}| > |\mathcal{F}|$ by hypothesis, and since $\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_{|\mathcal{M}|}$ are distinct and pairwise vertex-disjoint, there must exist at least one path p_i that goes from $\mathcal{S}$ to $\mathcal{T}$ avoiding $\mathcal{F}$. It is therefore sufficient to find such a path $\mathsf{p}_i = v_0 v_1 \cdots v_k$ by direct inspection. At that point, it is possible to reconstruct a path p going from S to v_0 (because $v_0 \in \mathcal{S}$), as well as a path going from v_k to T (because $v_k \in \mathcal{T}$). As already mentioned, in practice, the reconstruction can be implemented by maintaining a `map` that associates to every vertex v (eventually visited during the BFSs) the parent vertex that had led to discover v first. Then, $\langle \texttt{YES}, \mathsf{p} \rangle$ is returned at line 10.

2. If $|\mathcal{M}| \leq |\mathcal{F}|$, then the `compression_phase()` aims to *compress* the size of $\mathcal{T}$

down to $|\mathcal{T}'| \leq |\mathcal{F}|\, d_{S,T}$ as follows. Notice that in this case $\mathcal{M}$ is a maximum cardinality matching of $\mathcal{G}$, because $|\mathcal{M}| \leq |\mathcal{F}|$. So, the algorithm computes a minimum cardinality vertex-cover $\mathcal{X}$ of $\mathcal{G}$ at line 11, whose size is $|\mathcal{M}|$ by König's theorem [5]. The algorithm then proceeds at line 12 by considering the set $\mathcal{X}_\mathcal{S} = \mathcal{X} \cap \mathcal{S}$ (resp. $\mathcal{X}_\mathcal{T} = \mathcal{X} \cap \mathcal{T}$) of all vertices that lie both in the vertex-cover $\mathcal{X}$ and in the frontier set $\mathcal{S}$ (resp. $\mathcal{T}$). Here, it is crucial to notice that both $|\mathcal{X}_\mathcal{S}| \leq |\mathcal{F}|$ and $|\mathcal{X}_\mathcal{T}| \leq |\mathcal{F}|$ hold, because $|\mathcal{X}| = |\mathcal{M}| \leq |\mathcal{F}|$. The fact that, since $\mathcal{X}$ is a vertex-cover of $\mathcal{G}$, any directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}$ to $\mathcal{T}$ must go either from $\mathcal{X}_\mathcal{S}$ to $\mathcal{T}$ or from $\mathcal{S} \setminus \mathcal{X}_\mathcal{S}$ to $\mathcal{X}_\mathcal{T}$ plays a pivotal role. Stated otherwise, there exists no directed path in $\mathcal{H}_n$ that goes from $\mathcal{S} \setminus \mathcal{X}_\mathcal{S}$ to $\mathcal{T} \setminus \mathcal{X}_\mathcal{T}$, simply because $\mathcal{X}$ is a vertex cover of $\mathcal{G}$. At that point, the compression $\mathcal{T}'$ gets enriched with $\mathcal{X}_\mathcal{T}$ at line 13.

Then, `compression_phase()` seeks a directed path in $\mathcal{H}_n$ that eventually goes from $\mathcal{X}_S$ to $\mathcal{T}$. This is done at line 14 by running `double-bfs_phase()` on $\langle \mathcal{X}_\mathcal{S}, \mathcal{T}, \mathcal{F}, \ell_\uparrow, \ell_\downarrow, d_{S,T}, n \rangle$. Since $|\mathcal{X}_\mathcal{S}| \leq |\mathcal{F}|$, that execution results into an update of both the frontier set $\mathcal{S}$ and of its level counter $\ell_\uparrow$. Let $\mathcal{S}^{(i+1)}$ be the updated value of $\mathcal{S}$ and let $\ell_\uparrow^{(i+1)}$ be that of $\ell_\uparrow$. Note that, since $|\mathcal{T}| > |\mathcal{F}|\, d_{S,T}$ holds as a pre-condition of `compression_phase()`, neither $\mathcal{T}$ nor $\ell_\downarrow$ are ever updated at line 14. Upon completion of this supplementary `double-bfs_phase()`, if $\mathcal{S}^{(i+1)} = \emptyset$ or both $\ell_\uparrow^{(i+1)} + \ell_\downarrow = d_{S,T}$ *and* $\mathcal{S}^{(i+1)} \cap \mathcal{T} = \emptyset$ at line 15, then $\mathcal{T}'$ is returned at line 16 of `compression_phase()`.

Otherwise, if $\ell_\uparrow^{(i+1)} + \ell_\downarrow = d_{S,T}$ *and* $\mathcal{S}^{(i+1)} \cap \mathcal{T} \neq \emptyset$ at line 17, the sought directed path `p` in $\mathcal{H}_n$ that goes from source S to target T avoiding $\mathcal{F}$ can be reconstructed from $\mathcal{S}^{(i+1)}$ and $\mathcal{T}$ at line 18, so that `compression_phase()` returns $\langle$`YES`, `p`$\rangle$ and halts soon after at line 19.

Otherwise, if $\mathcal{S}^{(i+1)} \neq \emptyset$ and $\ell_\uparrow^{(i+1)} + \ell_\downarrow < d_{S,T}$, the next iteration will run on the novel frontier set $\mathcal{S}^{(i+1)}$ and its updated level counter $\ell_\uparrow^{(i+1)}$. It is not difficult to prove[5] that each iteration increases $\ell_\uparrow$ by at least one unit, so that the `while-loop` at line 2 of `compression_phase()` can be iterated at most $d_{S,T}$ times overall. In particular, this fact implies that $|\mathcal{T}'| \leq |\mathcal{F}|\, d_{S,T}$ always holds at line 16 of `compression_phase()`.

Fig. 2 illustrates the family of all frontier sets considered throughout `compression_phase()`, where the following notation is assumed: $\max_i$ is the total number of iterations of the `while-loop` at line 2 of `compression_phase()`, $\mathcal{X}^{(i)}$ is the vertex-cover computed at the i^{th} iteration of line 11, $\mathcal{X}_\mathcal{S}^{(i)}$ and $\mathcal{X}_\mathcal{T}^{(i)}$ are the sets computed at the i^{th} iteration of line 12, and $\mathcal{S}^{(i)}$ is the frontier set computed at the i^{th} iteration of line 14. The compression of $\mathcal{T}$ (possibly returned at line 16) is $\mathcal{T}' = \bigcup_{i=1}^{\max_i} \mathcal{X}_\mathcal{T}^{(i)}$.

3.3 A Remark On Decision Versus Search

Algorithm 1 tackles the Search-Task of Hy-stCon. If we merely want to answer the Decision-Task instead, we can simplify the algorithm by immediately returning `YES` if $|\mathcal{M}| > |\mathcal{F}|$ at line 5 of `compression_phase()`. This is because in that case, Theorem 2 guarantees the existence of a family of $|\mathcal{M}| > |\mathcal{F}|$ vertex-disjoint paths

[5] See Appendix B.2

in $\mathcal{H}_n$ that go from the current source frontier $\mathcal{S}$ to the target frontier $\mathcal{T}$, which suffices to conclude that at least one of those paths avoids $\mathcal{F}$. This simplification improves the time complexity of our algorithm for solving the DECISION-TASK by a polynomial factor over that for the SEARCH-TASK.

4 Conclusion

With the intention of integrating more biologically relevant constraints into classical genome rearrangement problems, we introduced in this paper the GUIDED SORTING problem. We broadly define it as the problem of transforming two genomes into one another using as few operations as possible from a given fixed set of allowed operations while avoiding a set of nonviable genomes. We gave a polynomial time algorithm for solving this problem in the case where genomes are represented by permutations, under the assumptions that 1) permutations can only be modified by exchanging any two elements, 2) the sequence to seek must be optimal, and 3) the permutation to sort is an involution.

Many questions remain open, most notably that of the computational complexity of the GUIDED SORTING problem, whether under assumptions (1) and (2) or in a more general setting (i.e., using structures other than permutations, operations other than exchanges, or allowing sequences to be "as short as possible" instead of optimal). One could also investigate "implicit" representations for the set of forbidden intermediate permutations, e.g. all permutations that avoid a given (set of) pattern(s), or that belong to a specific conjugacy class. Aside from complexity issues, future work shall also focus on extending the approach we proposed to other families of instances of the GUIDED SORTING problem, and identifying other tractable (or intractable) cases or variants of it; for instance, we plan to extend our algorithmic results to the family of graphs satisfying the *shadow-matching* [17] condition.

References

[1] S. BÉRARD, A. BERGERON, AND C. CHAUVE, *Conservation of combinatorial structures in evolution scenarios*, in RECOMB'04, vol. 3388 of LNCS, Springer, 2004, pp. 16–19.

[2] A. BERGERON, M. BLANCHETTE, A. CHATEAU, AND C. CHAUVE, *Reconstructing ancestral gene orders using conserved intervals*, in WABI'04, vol. 3240 of LNCS, Springer, 2004, pp. 14–25.

[3] L. CUÉNOT, *Les races pures et leurs combinaisons chez les souris*, Archives de Zoologie Experimentale, 3 (1905), pp. cxxiii–cxxxii.

[4] M. DEZA AND T. HUANG, *Metrics on permutations, a survey*, Journal of Combinatorics, Information and System Sciences, 23 (1998), pp. 173–185.

[5] R. DIESTEL, *Graph Theory (Graduate Texts in Mathematics)*, Springer, Aug. 2005.

[6] G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Vialette, *Combinatorics of Genome Rearrangements*, The MIT Press, 2009.

[7] M. Figeac and J.-S. Varré, *Sorting by reversals with common intervals*, in WABI'04, vol. 3240 of LNCS, Springer, 2004, pp. 26–37.

[8] H. Gabow, S. Maheshwari, and L. Osterweil, *On two problems in the generation of program test paths*, IEEE Trans. Software Eng., (1976), pp. 227–231.

[9] S. Gluecksohn-Waelsch, *Lethal genes and analysis of differentiation*, Science, 142 (1963), pp. 1269–1276.

[10] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky, *Testing monotonicity*, Combinatorica, 20 (2000), pp. 301–337.

[11] J. Hopcroft and R. Karp, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM Journal on Computing, 2 (1973), pp. 225–231.

[12] D. E. Knuth, *The Art of Computer Programming, Volume III: Sorting and Searching*, Addison-Wesley, 1973.

[13] K. Krause, R. Smith, and M. Goodwin, *Optimal software test planning through automated network analysis*, in Proceedings 1973 IEEE Symposium Computer Software Reliability, IEEE, 1973, pp. 18–22.

[14] A. Labarre, *Lower bounding edit distances between permutations*, SIAM Journal on Discrete Mathematics, 27 (2013), pp. 1410–1428.

[15] S. Lakshmivarahan, J.-S. Jwo, and S. K. Dhall, *Symmetry in interconnection networks based on Cayley graphs of permutation groups: A survey*, Parallel Computing, 19 (1993), pp. 361–407.

[16] E. Lehman and D. Ron, *On disjoint chains of subsets*, Journal of Combinatorial Theory, Series A, 94 (2001), pp. 399–404.

[17] M. Logan and S. Shahriari, *A new matching property for posets and existence of disjoint chains*, Journal of Combinat. Theory, Series A, (2004).

[18] H. Yinnone, *On path avoiding forbidden pairs of vertices in a graph*, Discrete Applied Mathematics, 74 (1997), pp. 85–92.

A Appendix: Instances of GUIDED SORTING that Reduce to HY-STCON

We examine in this section specific instances of GUIDED SORTING which can be solved through a reduction to HY-STCON. We say that permutations that may occur in an optimal sorting sequence for a given permutation π are *relevant*, and all others are *irrelevant*. The distinction will matter when sorting a particular permutation since, as we shall see, the structure of π (however it is measured) will have implications on that of relevant permutations and will allow us to simplify the set of forbidden permutations by discarding irrelevant ones. For a fixed set S of operations, we let $R_S(\pi)$ denote the set of permutations that are relevant to π. Undefined terms and unproven properties of permutations below are well-known, and details can be found in standard references such as [12].

A.1 GUIDED SORTING For Exchanges

Recall that every permutation π in $\mathfrak{S}_n$ decomposes in a single way into *disjoint cycles* (up to the ordering of cycles and of elements within each cycle). This decomposition corresponds to the cycle decomposition of the directed graph $G(\pi) = (V, A)$, where $V = [n]$ and $A = \{(i, \pi_i) \mid 1 \leq i \leq n\}$. The *length* of a cycle of π is then simply the number of elements it contains, and the number of cycles of π is denoted by $c(\pi)$.

The *Cayley distance* of a permutation π is the length of an optimal sorting sequence of exchanges for π, and its value is $n - c(\pi)$. Therefore, when searching for an optimal sorting sequence, we may restrict our attention to exchanges that split a particular cycle into two smaller ones.

Let $(\pi, \mathcal{F}, S, K)$ be an instance of GUIDED SORTING such that S is the set of all exchangess and where the permutation π to sort is an *involution*, i.e. a permutation whose cycles have length at most two. It is customary to omit cycles of length 1, and to write a permutation $\pi = \langle \pi_1 \ \pi_2 \ \cdots \ \pi_n \rangle$ with k cycles of length 2 as $c_1 c_2 \cdots c_k$. Since we are looking for an optimal sorting sequence, we may assume that all permutations in $\mathcal{F}$ are relevant, which in this case means that every permutation ϕ in $\mathcal{F}$ is an involution and its 2-cycles form a proper subset of those of π. Our instance of GUIDED SORTING then translates to the following instance of HY-STCON:

- $\pi \mapsto [k]$ in the following way: $c_i \mapsto i$ for $1 \leq i \leq k$;
- each permutation ϕ in $\mathcal{F}$ is mapped onto a subset of $[k]$ by replacing its cycles with the indices obtained in the first step; we let $\mathcal{F}'$ denote the collection of subsets of $[k]$ obtained by applying that mapping to each ϕ in $\mathcal{F}$.

The resulting HY-STCON instance is then $\langle [k], \emptyset, \mathcal{F}', k \rangle$, and a solution to instance $(\pi, \mathcal{F}, S, K)$ of GUIDED SORTING exists if and only if a solution to instance $\langle [k], \emptyset, \mathcal{F}', k \rangle$ of HY-STCON exists; the translation of the solution from the latter formulation to the former is straightforward.

A.2 GUIDED SORTING For Adjacent Exchanges

Recall that an *inversion* in a permutation π in $\mathfrak{S}_n$ is a pair (π_i, π_j) with $1 \leq i < j \leq n$ and $\pi_i > \pi_j$. Let $(\pi, \mathcal{F}, S, K)$ be an instance of GUIDED SORTING where S is the

set of all *adjacent* exchanges, i.e. exchanges that act on consecutive positions. It is well-known that in this case, any optimal sorting sequence for π has length equal to the number of inversions of π, which means that in the search for an optimal sorting sequence, we may restrict our attention to adjacent exchanges that act on inversions that consist of adjacent elements and disregard *fixed points* (i.e. positions i such that $\pi_i = i$, which correspond to elements that are already where they should be in ι).

Let us now assume that all k inversions of π are made of adjacent elements, and denote $\pi = i_1 i_2 \cdots i_k$, where each i_j is an inversion. Since we are looking for an optimal sorting sequence, we may assume that all permutations in $\mathcal{F}$ are relevant, which in this case means that all inversions of any permutation ϕ in $\mathcal{F}$ form a proper subset of those of π. The reduction to Hy-stCon in that setting is very similar to that given in the case of exchanges:

- $\pi \mapsto [k]$ in the following way: $i_j \mapsto j$ for $1 \leq j \leq k$;
- each permutation ϕ in $\mathcal{F}$ is mapped onto a subset of $[k]$ by replacing its inversions with the indices obtained in the first step; we let $\mathcal{F}'$ denote the collection of subsets of $[k]$ obtained by applying that mapping to each ϕ in $\mathcal{F}$.

The resulting Hy-stCon instance is then $\langle [k], \emptyset, \mathcal{F}', k \rangle$, and a solution to instance $(\pi, \mathcal{F}, S, K)$ of guided sorting exists if and only if a solution to instance $\langle [k], \emptyset, \mathcal{F}', k \rangle$ of Hy-stCon exists; the translation of the solution from the latter formulation to the former is straightforward.

B Appendix: Omitted Proofs

B.1 The Proof of Theorem 1

Our proof of Theorem 1 relies on certain connectivity properties of hypercube graphs, and in particular the existence of a family of certain vertex-disjoint paths in $\mathcal{H}_n$ that we call *"Lehman-Ron paths"*, which is guaranteed by Theorem 2.

Although Theorem 2 was initially proved and applied in the specific area of testing monotonicity [10], it is of independent interest and related results could be useful in the context of packet routing on the hypercube network. Lehman and Ron provided an elegant inductive proof of that result [16]. In the present work, we point out that a careful analysis of their proof allows us to "extract" a simple recursive algorithm for computing all Lehman-Ron paths in polynomial time. We now describe that algorithm, whose correctness follows from the arguments used by Lehman and Ron in their original proof of Theorem 2 (see [16] for more details). Its time complexity can be derived by taking into account Hopcroft-Karp's algorithm for computing maximum cardinality matchings in bipartite graphs [11], and will be analyzed in details at the end of this section.

The algorithm we describe is named `compute_Lehman-Ron_paths()`. The intuition underlying it is simply to follow the structure of Lehman and Ron's proof of Theorem 2 and to analyze it from the algorithmic standpoint. Its pseudocode is given below.

Algorithm 4: computing Lehman-Ron's paths.

Procedure *compute_Lehman-Ron_paths*$(\mathcal{R}, \mathcal{S}, \varphi, n)$

Input: a Lehman-Ron tuple $\langle \mathcal{R}, \mathcal{S}, \varphi, n \rangle$.

Output: a family of m vertex-disjoint directed paths $\mathsf{p}_1, \ldots, \mathsf{p}_m$ in $\mathcal{H}_n$ such that $\mathcal{R} \cup \mathcal{S} \subseteq \bigcup_{i=1}^{m} \mathsf{p}_i$.

1 **if** $s = r + 1$ **then**

2 **return** *compute_paths_from_bijection*$(\mathcal{S}, \varphi, n)$;

3 $m \leftarrow |\mathcal{S}|$; // assume $|\mathcal{S}| = |\mathcal{R}|$

4 $\mathcal{Q} \leftarrow$ `compute_Q`$(\mathcal{S})$;

5 $\mathcal{K} \leftarrow$ `compute_auxiliary_network`$(\mathcal{R}, \mathcal{Q}, \mathcal{S})$;

6 $\langle \mathsf{p}'_1, \mathsf{p}'_2, \ldots, \mathsf{p}'_m \rangle \leftarrow$ `compute_vertex_disjoint_paths`$(\mathcal{K})$;

7 $\langle \mathcal{Q}', \varphi', \varphi'' \rangle \leftarrow$ `compute_auxiliary_bijections`$(\langle \mathsf{p}'_1, \mathsf{p}'_2, \ldots, \mathsf{p}'_m \rangle, m)$

8 $\langle \mathsf{p}''_1, \mathsf{p}''_2, \ldots, \mathsf{p}''_m \rangle \leftarrow$ `compute_Lehman-Ron_paths`$(\mathcal{R}, \mathcal{Q}', (\varphi')^{-1}, n)$;

9 $\langle \mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_m \rangle \leftarrow$ `extend_paths`$(\langle p''_1, \mathsf{p}''_2, \ldots, p''_m \rangle, \mathcal{Q}', \varphi'', m)$;

10 **return** $\langle \mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_m \rangle$;

The algorithm takes as input a Lehman-Ron tuple $\langle \mathcal{R}, \mathcal{S}, \varphi, n \rangle$, and outputs a family $\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_m$ of Lehman-Ron paths joining $\mathcal{R}$ to $\mathcal{S}$. Recall that Lehman-Ron tuples satisfy the following properties:

1. the families of sets $\mathcal{R} \subseteq \mathcal{H}_n^{(r)}$ and $\mathcal{S} \subseteq \mathcal{H}_n^{(s)}$ are such that $|\mathcal{S}| = |\mathcal{R}| = m$,
2. r, s and $n \in \mathbb{N}$ are such that $0 \leq r < s \leq n$, and
3. $\varphi : \mathcal{S} \to \mathcal{R}$ is a bijection such that $\forall\ S \in \mathcal{S} : \varphi(S) \subset S$.

As a base case of the algorithm, if $s = r + 1$ (line 1), then the sought family of directed paths $\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_m$ is simply a set of m pairwise vertex-disjoint arcs oriented from $\mathcal{S}$ to $\mathcal{R}$, which are already given by the input bijection φ (line 2).

We now focus on the general case $s > r + 1$. To begin with, we introduce the following proposition, which was already implicit in [16] and which is actually a straightforward consequence of Theorem 2.

Proposition 1. *[16] Given $n, m \in \mathbb{N}$, consider two families of sets $\mathcal{R} \subseteq \mathcal{H}_n^{(r)}$ and $\mathcal{S} \subseteq \mathcal{H}_n^{(s)}$ where $|\mathcal{R}| = |\mathcal{S}| = m$ and $0 \leq r < s \leq n$. Let $\mathcal{Q}$ (resp. $\mathcal{P}$) be the set of vertices in $\mathcal{H}_n^{(s-1)}$ (resp. $\mathcal{H}_n^{(r+1)}$) that lie on any directed path from some vertex in $\mathcal{R}$ to some vertex in $\mathcal{S}$. Then, $|\mathcal{Q}| \geq m$ and $|\mathcal{P}| \geq m$.*

The algorithm first computes the set $\mathcal{Q}$ of all vertices in $\mathcal{H}_n^{(s-1)}$ that lie on any directed path from some vertex in $\mathcal{R}$ to some vertex in $\mathcal{S}$. This step is encoded by `compute_Q()` (line 4). The algorithm then invokes (at line 5) a procedure called `compute_auxiliary_network()`, which constructs a directed auxiliary network $\mathcal{K} = (V_{\mathcal{K}}, A_{\mathcal{K}})$ which will be useful in the following steps and is defined by:

- $V_{\mathcal{K}} = \{\mathbf{s}, \mathbf{t}\} \cup \mathcal{R} \cup \mathcal{S} \cup \mathcal{Q}$, where $\mathbf{s}$ (resp. $\mathbf{t}$) is an auxiliary source (resp. target) vertex, i.e. $\{\mathbf{s}, \mathbf{t}\} \cap (\mathcal{R} \cup \mathcal{Q} \cup \mathcal{S}) = \emptyset$;
- $A_{\mathcal{K}}$ is defined as follows:
  - the source vertex $\mathbf{s}$ is joined to every vertex in $\mathcal{R}$;
  - for each $R \in \mathcal{R}$ and $Q \in \mathcal{Q}$, R is joined to Q if and only if $R \subset Q$;
  - similarly, for each $Q \in \mathcal{Q}$ and $S \in \mathcal{S}$, Q is joined to S if and only if $Q \subset S$;
  - finally, every vertex in $\mathcal{S}$ is joined to $\mathbf{t}$.

We remark that, as shown in [16], the following proposition holds on $\mathcal{K}$.

Proposition 2. *[16] The minimum $(\boldsymbol{s}, \boldsymbol{t})$-vertex-separator of $\mathcal{K}$ has size m.*

As a corollary, and by applying Menger's vertex-connectivity theorem (which is recalled below), the existence of m internally-vertex-disjoint directed $(\mathbf{s}, \mathbf{t})$-paths, denoted $\mathsf{p}'_1, \mathsf{p}'_2, \ldots, \mathsf{p}'_m$, is thus guaranteed.

Theorem 4 (Menger [5])**.** *Let $G = (V, A)$ be a directed graph, and let u and v be nonadjacent vertices in V. Then the maximum number of internally-vertex-disjoint directed (u, v)-paths in G equals the minimum number of vertices from $V \setminus \{u, v\}$ whose deletion destroys all directed (u, v)-paths in G.*

B.1.1 How to compute $\mathsf{p}'_1, \mathsf{p}'_2, \ldots, \mathsf{p}'_m$.

We argue that it is possible to compute efficiently the family of directed paths $\mathsf{p}'_1, \mathsf{p}'_2, \ldots, \mathsf{p}'_m$ in $\mathcal{K}$ by finding a maximum cardinality matching in an auxiliary, *undirected* bipartite graph $\mathcal{K}'$. This reduction is performed by `compute_vertex_disjoint_paths()` at line 6. The undirected graph $\mathcal{K}' = (V_{\mathcal{K}'}, E_{\mathcal{K}'})$ is obtained from the directed graph $\mathcal{K}$ as follows: first, the set family $\mathcal{Q}$ gets split into two (disjoint) twin set families $\mathcal{Q}^{(\mathbf{in})}$ and $\mathcal{Q}^{(\mathbf{out})}$,i.e. $\mathcal{Q}^{(\mathbf{in})} = \{Q^{(\mathbf{in})} \mid Q \in \mathcal{Q}\}$ and $\mathcal{Q}^{(\mathbf{out})} = \{Q^{(\mathbf{out})} \mid Q \in \mathcal{Q}\}$ where $\mathcal{Q}^{(\mathbf{in})} \cap \mathcal{Q}^{(\mathbf{out})} = \emptyset$ and $|\mathcal{Q}^{(\mathbf{in})}| = |\mathcal{Q}^{(\mathbf{out})}| = |\mathcal{Q}|$. Thus, the vertex set of $\mathcal{K}'$ is:

$$V_{\mathcal{K}'} = \mathcal{R} \cup \mathcal{Q}^{(\mathbf{in})} \cup \mathcal{Q}^{(\mathbf{out})} \cup \mathcal{S}.$$

The edge set $E_{\mathcal{K}'}$ is obtained as follows:

- for each $R \in \mathcal{R}$ and $Q \in \mathcal{Q}$, R is joined to $Q^{(\mathbf{in})}$ if and only if $R \subset Q$;
- similarly, for each $Q \in \mathcal{Q}$ and $S \in \mathcal{S}$, $Q^{(\mathbf{out})}$ is joined to S if and only if $Q \subset S$;
- finally, $Q^{(\mathbf{in})}$ is joined to $Q^{(\mathbf{out})}$ for every $Q \in \mathcal{Q}$.

In the next proposition we derive some useful properties of $\mathcal{K}'$.

Proposition 3. *The graph $\mathcal{K}' = (V_{\mathcal{K}'}, E_{\mathcal{K}'})$, as defined above, is bipartite and it admits a perfect matching.*

Proof. The bipartiteness of $\mathcal{K}'$ follows from the bipartition $(\mathcal{R} \cup \mathcal{Q}^{(\mathbf{out})}, \mathcal{Q}^{(\mathbf{in})} \cup \mathcal{S})$. To see that $\mathcal{K}'$ admits a perfect matching, recall that, by Proposition 2 and by Theorem 4, there exist m internally-vertex-disjoint directed $(\mathbf{s}, \mathbf{t})$-paths $\mathsf{p}'_1, \mathsf{p}'_2, \ldots, \mathsf{p}'_m$ in $\mathcal{K}$. Then, for every $i \in [m]$, let $\mathsf{p}'_i = \mathbf{s}R_iQ_iS_i\mathbf{t}$ for some $R_i \in \mathcal{R}, Q_i \in \mathcal{Q}, S_i \in \mathcal{S}$. Finally, let us define $\hat{\mathcal{Q}} = \mathcal{Q} \setminus \{Q \mid \exists\, i \in [m] \text{ s.t. } \mathsf{p}'_i = \mathbf{s}R_iQS_i\mathbf{t}\}$. At this point, let us consider the following matching $\mathcal{M}$ of $\mathcal{K}'$:

$$\mathcal{M} = \big\{\{R_i, Q_i^{(\mathbf{in})}\}, \{Q_i^{(\mathbf{out})}, S_i\} \mid \exists\, i \in [m] \text{ s.t. } \mathsf{p}'_i = \mathbf{s}R_iQ_iS_i\mathbf{t}\big\} \cup \big\{\{Q^{(\mathbf{in})}, Q^{(\mathbf{out})}\} \mid Q \in \hat{\mathcal{Q}}\}\big\}.$$

Since $m = |\mathcal{R}| = |\mathcal{S}|$ and $\mathsf{p}'_1, \mathsf{p}'_2, \ldots, \mathsf{p}'_m$ are internally-vertex-disjoint, it follows that $\mathcal{M}$ is a perfect matching of $\mathcal{K}'$. □ □

We can now show how to compute $\mathsf{p}'_1, \mathsf{p}'_2, \ldots, \mathsf{p}'_m$ based on $\mathcal{K}$. Firstly, the procedure `compute_vertex_disjoint_paths()` constructs $\mathcal{K}'$ as explained above and computes a maximum cardinality matching $\mathcal{M}$ of $\mathcal{K}'$ (e.g. with Hopcroft-Karp's algorithm [11]), which is perfect by Proposition 3. Therefore, the following property holds: for every $Q \in \mathcal{Q}$, there exists $R \in \mathcal{R}$ such that $\{R, Q^{(\mathbf{in})}\} \in \mathcal{M}$ if and only if there exists $S \in \mathcal{S}$ such that $\{Q^{(\mathbf{out})}, S\} \in \mathcal{M}$. We can then proceed as follows: for each $R_i \in \mathcal{R}$, the algorithm finds $Q_i \in \mathcal{Q}$ such that $\{R_i, Q_i^{(\mathbf{in})}\} \in \mathcal{M}$ and then it finds $S_i \in \mathcal{S}$ such that $\{Q_i^{(\mathbf{out})}, S_i\} \in \mathcal{M}$. Then, `compute_vertex_disjoint_paths()` returns the family of paths $\mathsf{p}'_1, \mathsf{p}'_2, \ldots, \mathsf{p}'_m$ defined as: $\mathsf{p}'_i = \mathbf{s}R_iQ_iS_i\mathbf{t}$ for every $i \in [m]$. Since $\mathcal{M}$ is a perfect matching of $\mathcal{K}'$, the paths $\mathsf{p}'_1, \mathsf{p}'_2, \ldots, \mathsf{p}'_m$ are internally-vertex-disjoint.

Let $\mathcal{Q}' = \{Q \mid \exists\, i \in [m] \text{ s.t. } \mathsf{p}'_i = \mathbf{s}R_iQS_i\mathbf{t}\}$. Once we have computed $\mathsf{p}'_1, \mathsf{p}'_2, \ldots, \mathsf{p}'_m$, we can deduce two bijections that will be helpful in obtaining the wanted paths:

$$\varphi' : \mathcal{R} \to \mathcal{Q}' \text{ and } \varphi'' : \mathcal{Q}' \to \mathcal{S}.$$

The first bijection is defined for any $R \in \mathcal{R}$ as $\varphi'(R) = Q$ (where $Q \in \mathcal{Q}'$) provided there exists some p'_i joining R to Q; similarly, the second bijection is defined for any $Q \in \mathcal{Q}'$ as $\varphi''(Q) = S$ (where $S \in \mathcal{S}$) provided there exists some p'_i joining Q to S. These bijections are computed by `compute_auxiliary_bijections()` at line 7.

At this point, since the distance between $\mathcal{R}$ and $\mathcal{Q}'$ equals $s - 1$, a recursive call to `compute_Lehman-Ron_paths()` on input $\langle \mathcal{R}, \mathcal{Q}', (\varphi')^{-1}, n \rangle$ yields, at line 8, a family of Lehman-Ron paths $\mathsf{p}''_1, \mathsf{p}''_2, \ldots, \mathsf{p}''_m$ joining $\mathcal{R}$ to $\mathcal{Q}'$.

Indeed, we argue that it is possible to construct, starting from $\mathsf{p}''_1, \mathsf{p}''_2, \ldots, \mathsf{p}''_m$, the sought family of Lehman-Ron paths $\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_m$ that join $\mathcal{R}$ to $\mathcal{S}$. Actually, this can be done just by taking into account the bijection φ'': since φ'' joins $\mathcal{Q}'$ to $\mathcal{S}$, it suffices to perform the following steps in practice:

1. consider the last vertex Q_i of p''_i (i.e. the unique vertex $Q_i \in \mathcal{Q}'$ such that $Q_i \in \mathsf{p}''_i \cap \mathcal{Q}'$);
2. let $S_i = \varphi''(Q_i)$;
3. concatenate S_i at the end of p''_i (i.e. $\mathsf{p}_i = \mathsf{p}''_i\, S_i$).

This construction is performed by the `extend_paths()` procedure at line 9. Since $\mathsf{p}''_1, \mathsf{p}''_2, \ldots, \mathsf{p}''_m$ are vertex-disjoint and $\varphi'' : \mathcal{Q}' \to \mathcal{S}$ is a bijection, $\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_m$ is the sought family of Lehman-Ron paths joining $\mathcal{R}$ to $\mathcal{S}$.

B.1.2 Complexity Analysis (Proof of Theorem 3).

We now turn to the time complexity analysis of Algorithm 4, going through each line in details.

- line 2: `compute_paths_from_bijection()` (line 2) takes time at most $O(m)$, which corresponds to the time needed to inspect the input bijection φ.
- line 4: `compute_Q()` takes time at most $O(mn)$: for each $S \in \mathcal{S}$, the procedure inspects the predecessors $N^{\mathtt{in}}(S)$, and the time bound follows from the fact that $|\mathcal{S}| = m$ and $|N^{\mathtt{in}}(S)| \leq n$.
- line 5: we argue that $|V_{\mathcal{K}}| = O(mn)$ and $|A_{\mathcal{K}}| = O(m^2 n)$. Indeed, recall that $|\mathcal{R}| = |\mathcal{S}| = m$ by hypothesis; and since every vertex of $\mathcal{S}$ has at most n neighbours in $\mathcal{Q}$, we have $|\mathcal{Q}| \leq mn$. This in turn implies that $|V_{\mathcal{K}}| \leq 2+2m+mn$; moreover, each of the m vertices in $\mathcal{R}$ has at most mn neighbours, which all lie in $\mathcal{Q}$. Therefore, $|A_{\mathcal{K}}| \leq 2m + m^2 n + mn$, and the procedure `compute_auxiliary_network()` takes time at most $O(|V_{\mathcal{K}}|+|A_{\mathcal{K}}|) = O(m^2 n)$.
- line 6: `compute_vertex_disjoint_paths()` takes time at most $O\big(m^{5/2} n^{3/2}\big)$. Indeed, let us consider the auxiliary (undirected) bipartite graph $\mathcal{K}' = (V_{\mathcal{K}'}, E_{K'})$ defined above. Since $|V_{\mathcal{K}}| = O(mn)$ and $|A_{\mathcal{K}}| = O(m^2 n)$, we have $|V_{\mathcal{K}'}| = O(mn)$ and $|E_{\mathcal{K}'}| = O(m^2 n)$ by construction. A maximum cardinality matching $\mathcal{M}$ of $\mathcal{K}'$ can be computed with the Hopcroft-Karp's algorithm [11] within time $O(\sqrt{|V_{\mathcal{K}'}|}\,|E_{\mathcal{K}'}|) = O(m^{5/2} n^{3/2})$, which yields the claimed time bound.
- finally, lines 7 (`compute_auxiliary_bijections()`) and 9 (`extend_paths()`) take time at most $O(m)$.

To obtain the total time complexity of `compute_Lehman-Ron_paths()`, it is sufficient to observe that the depth of the recursion stack (originating from line 8) equals the distance $d = s - r$ between the families of sets that were originally given as input, $\mathcal{R}$ and $\mathcal{S}$, and that the most expensive computation at each step of the recursion is clearly the maximum cardinality matching computation that is performed on the auxiliary bipartite graph $\mathcal{K}'$. Therefore, we conclude that the worst-case time complexity of `compute_Lehman-Ron_paths()` is $O\big(m^{5/2} n^{3/2} d\big)$.

B.2 Correctness Analysis of Algorithm 1

The present subsection aims to show that the procedure `solve_Hy-stCon()` is correct. A formal statement of that is provided in the next theorem.

Theorem 5. *Let $\mathcal{I} = \langle S, T, \mathcal{F}, n\rangle$ be any instance of* Hy-stCon*. Given $\mathcal{I}$ as input, the procedure `solve_Hy-stCon()` halts within a finite number of steps. Moreover, it returns as output a directed path `p` in $\mathcal{H}_n$ that goes from source S to target T avoiding $\mathcal{F}$, provided that at least one such path exists; otherwise, the output is simply `NO`.*

We are going to show a sequence of results that shall ultimately lead us to prove Theorem 5. Hereafter, it is assumed that $\langle S, T, \mathcal{F}, n\rangle$ is an instance (of Hy-stCon) given as input to the `solve_Hy-stCon()` procedure. Lemmas 1 to 3 below show that procedures `double-bfs_phase()` and `compression_phase()`, which are called by `solve_Hy-stCon()`, halt within a finite number of steps.

Lemma 1. *Any invocation of `double-bfs_phase()` halts within a finite number of steps. In particular, the `while-loop` at line 1 of the `bfs_phase()` iterates at most $d_{S,T}$ times.*

Proof. Consider the `while-loop` at line 1 of `bfs_phase()`. At each iteration of line 3, the level counter ℓ_x gets incremented. Notice that this is the only line at which ℓ_x may be modified, and also notice that ℓ_y is never modified. Therefore, $\ell_x + \ell_y$ can only increase and not decrease. Since the `while-loop` at line 1 of `bfs_phase()` halts as soon as $\ell_x + \ell_y = d_{S,T}$, the thesis follows. □ □

Lemma 2. *Each iteration of the `while-loop` at line 2 of `compression_phase()` increases $\ell_\uparrow + \ell_\downarrow$ by at least one unit, either until $\ell_\uparrow + \ell_\downarrow = d_{S,T}$ or until the procedure halts by reaching either line 10, line 16 or line 19.*

Proof. Consider any iteration of the `while-loop` at line 2 of `compression_phase()`. Let $\mathcal{G}$ be the bipartite graph computed at line 3, and let $\mathcal{M}$ be the matching of $\mathcal{G}$ computed at line 4. If $|\mathcal{M}| > |\mathcal{F}|$, then line 10 gets executed, so the procedure halts within a finite number of steps by virtue of our discussion in Appendix B.1. Otherwise $|\mathcal{M}| \leq |\mathcal{F}|$. Recall that, since $|\mathcal{M}| \leq |\mathcal{F}|$, then $\mathcal{M}$ is a maximum matching of $\mathcal{G}$; also recall that $\mathcal{X}_\mathcal{S} = \mathcal{X} \cap \mathcal{S}$ where $\mathcal{X}$ is a minimum vertex cover of $\mathcal{G}$ (line 12). Since $|\mathcal{X}| = |\mathcal{M}|$, then $|\mathcal{X}_\mathcal{S}| \leq |\mathcal{X}| = |\mathcal{M}| \leq |\mathcal{F}|$. Moreover, since $|\mathcal{M}| \leq |\mathcal{F}|$, `double-bfs_phase()` gets invoked at line 14 on input $\langle \mathcal{X}_\mathcal{S}, \mathcal{T}, \mathcal{F}, \ell_\downarrow, \ell_\uparrow, d_{S,T}, n\rangle$ and halts within a finite number of steps by Lemma 1. Let us analyze its behavior with respect to $\mathcal{X}_\mathcal{S}$. If $\mathcal{X}_\mathcal{S} = \emptyset$, then `double-bfs_phase()` returns an empty frontier set $\mathcal{S}$ as output, which leads to the termination of `compression_phase()` at line 16. Moreover, if $\ell_\uparrow + \ell_\downarrow = d_{S,T}$, then `compression_phase()` halts either at line 16 or at line 19. Otherwise, we must have $1 \leq |\mathcal{X}_\mathcal{S}| \leq |\mathcal{F}|$ and $\ell_\uparrow + \ell_\downarrow < d_{S,T}$, in that case the condition for entering the `while-loop` at line 1 of the `bfs_phase()` is satisfied; therefore, at line 3 of `bfs_phase()`, the level counter $\ell_\uparrow$ gets incremented. This implies the thesis. □ □

Lemma 3. *Any invocation of `compression_phase()` halts within a finite number of steps. In particular, the `while-loop` at line 2 of the `compression_phase()` iterates at most $d_{S,T}$ times.*

Proof. Firstly, recall Lemma 2. Then, notice that as soon as $\ell_\uparrow + \ell_\downarrow = d_{S,T}$ the `compression_phase()` then halts either at line 16 (if $\mathcal{S} \cap \mathcal{T} = \emptyset$) or at line 19 (if $\mathcal{S} \cap \mathcal{T} \neq \emptyset$). This implies that the `while-loop` at line 2 of `compression_phase()` iterates at most $d_{S,T}$ times. □ □

We now prove some useful properties of `compression_phase()` and `solve_HystCon()`.

Lemma 4. *The following invariant is maintained at each line of `solve_HystCon()` and at each line of `compression_phase()`. For every $S' \in \mathcal{S}$ there exists a directed path in $\mathcal{H}_n$ that goes from S to S' avoiding $\mathcal{F}$; similarly, for every $T' \in \mathcal{T}$ there is a directed path in $\mathcal{H}_n$ that goes from T' to T avoiding $\mathcal{F}$.*

Proof. At the beginning of the procedure $\mathcal{S} = \{S\}$ and $\mathcal{T} = \{T\}$, so the thesis holds. At each subsequent step, the only way in which a novel vertex can be added either to $\mathcal{S}$ or $\mathcal{T}$ is by invoking the `double_bfs_phase()`, which preserves connectivity and avoids $\mathcal{F}$ by construction at line 3 of `next_step_bfs()`. □ □

Lemma 5. *Assume that any invocation of `compression_phase()` halts by returning $\langle$`YES`$, \mathsf{p}\rangle$. Then p is a directed path in $\mathcal{H}_n$ that goes from source S to target T avoiding $\mathcal{F}$.*

Proof. If `compression_phase()` returns `p` as output, then the last iteration of the `while-loop` at line 2 must reach either line 10 or line 19:

1. Assume that line 10 is reached at the last iteration. Then, during that iteration, the matching $\mathcal{M}$ (computed at line 4 on input $\mathcal{G}$) has size $|\mathcal{M}| > |\mathcal{F}|$. Recall that $\mathcal{G}$ is a bipartite graph on bipartition $(\mathcal{S}, \mathcal{T})$. Let $\mathcal{M}_\mathcal{S}$ (resp. $\mathcal{M}_\mathcal{T}$ be the subset of all vertices in $\mathcal{S}$ (resp. $\mathcal{T}$) that belong to some edge in $\mathcal{M}$. Then, by Theorem 2, there exist $|\mathcal{M}|$ vertex-disjoint directed paths in $\mathcal{H}_n$, say $\mathsf{p}_1, \mathsf{p}_2, \ldots, \mathsf{p}_{|\mathcal{M}|}$, whose union contains all the vertices in $\mathcal{M}_\mathcal{S}$ and $\mathcal{M}_\mathcal{T}$. Since $|\mathcal{M}| > |\mathcal{F}|$, at least one of those paths — say, $\mathsf{p}_i = v_0 \cdots v_k$ — must avoid $\mathcal{F}$. By Proposition 4, the procedure `reconstruct_path()` (invoked at line 9) is able to compute a directed path p_{S,v_0} in $\mathcal{H}_n$ that goes from S to v_0 avoiding $\mathcal{F}$ (because $v_0 \in \mathcal{S}$, being the first step of p_i), and it is also able to compute a directed path $\mathsf{p}_{v_k,T}$ that goes from v_k to T avoiding $\mathcal{F}$ (because $v_k \in \mathcal{T}$, being the last step of p_i). Let $\mathsf{p} = \mathsf{p}_{S,v_0}\mathsf{p}_i\mathsf{p}_{v_k,T}$ be the directed path obtained by concatenation. `compression_phase()` then returns `p` at line 10.

2. Assume that line 19 is reached at the last iteration. Then, at that iteration, the condition checked at line 17 of `compression_phase()` must be satisfied; that is, we have $\ell_\uparrow + \ell_\downarrow = d_{S,T}$ and $\mathcal{S} \cap \mathcal{T} \neq \emptyset$. Let X be an arbitrary vertex in $\mathcal{S} \cap \mathcal{T}$. By Lemma 4, there exists at least one directed path $\mathsf{p}_{S,X}$ in $\mathcal{H}_n$ that goes from S to X avoiding $\mathcal{F}$ (because $X \in \mathcal{S}$); similarly, there exists at least one directed path $p_{X,T}$ in $\mathcal{H}_n$ that goes from X to T avoiding $\mathcal{F}$ (because $X \in \mathcal{T}$). Therefore, during that iteration, the procedure `reconstruct_path()` (invoked at line 18) is able to compute a path $\mathsf{p} = \mathsf{p}_{S,X}\mathsf{p}_{X,T}$ that goes from S to X, and then from X to T, which is the result returned by `compression_phase()` at line 19. □

□

The following result shows two useful properties of the frontier set returned by `compression_phase()`, for which we will need additional notation. Denote by $\max_i$ be the number of times that the `while-loop` at line 2 gets iterated throughout the whole execution of the `compression_phase()`.

Also, let us introduce the following notation, for each index $i \in [\max_i]$:

- let $\mathcal{X}^{(i)}$ be the vertex cover that is computed during the i-th iteration of line 11;
- let $\mathcal{X}_{\mathcal{S}}^{(i)}$ and $\mathcal{X}_{\mathcal{T}}^{(i)}$ be the sets computed during the i-th iteration of line 12;
- let $\mathcal{S}^{(i)}$ be the novel frontier set that is computed during the i-th iteration of line 14;

Moreover, we assume the notation $\mathcal{S}^{(0)} = \mathcal{S}$, so that $\mathcal{X}_{\mathcal{S}}^{(i)} = \mathcal{S}^{(i-1)} \cap \mathcal{X}^{(i)}$ holds for each iteration $i \in [\max_i]$. Notice that, since $|\mathcal{T}| > |\mathcal{F}|\, d_{S,T}$ holds by hypothesis, then $\mathcal{T}$ is not modified, at line 14, by the invocation of `double-bfs_phase()`. Indeed, $\mathcal{T}$ is never modified throughout the `compression_phase()`. Nevertheless, a novel set $\mathcal{T}' \subset \mathcal{T}$ gets constructed and possibly returned.

Proposition 4. *Assume that the procedure `compression_phase()` is invoked on input $\langle \mathcal{S}, \mathcal{T}, \mathcal{F}, \ell_\uparrow, \ell_\downarrow, d_{S,T}, n \rangle$, where $|\mathcal{T}| > |\mathcal{F}|\, d_{S,T}$ is required to hold as a pre-condition. Also, assume that the procedure halts at line 16, returning a novel frontier set $\mathcal{T}' \subset \mathcal{T}$. Then, the following properties hold:*

1. *$|\mathcal{T}'| \leq |\mathcal{F}|\, d_{S,T}$;*
2. *if $\mathtt{p}$ is any directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}$ to $\mathcal{T}$ avoiding $\mathcal{F}$, then $\mathtt{p}$ goes from $\mathcal{S}$ to $\mathcal{T}'$.*

Proof. Firstly notice that, if an invocation of the `compression_phase()` halts at line 16 by returning a novel frontier set $\mathcal{T}' \subset \mathcal{T}$, this means that neither line 10 nor line 19 are ever reached throughout that invocation. In particular this implies that, at each iteration i of the `while-loop` at line 2, the maximum matching $\mathcal{M}^{(i)}$ (computed at line 4) has size $|\mathcal{M}^{(i)}| \leq |\mathcal{F}|$; this fact is assumed throughout the whole proof.

1. *Proof of (1).* At each iteration $i \in [\max_i]$, the minimum vertex cover $\mathcal{X}^{(i)}$ has size:
$$|\mathcal{X}^{(i)}| = |\mathcal{M}^{(i)}| \leq |\mathcal{F}|.$$
Since $\mathcal{X}_{\mathcal{T}}^{(i)} = \mathcal{X}^{(i)} \cap \mathcal{T}$ at line 12, then $|\mathcal{X}_{\mathcal{T}}^{(i)}| \leq |\mathcal{X}^{(i)}| \leq |\mathcal{F}|$. Moreover, recall that $\mathcal{T}'$ gets enriched by $\mathcal{X}^{(i)}$ at each iteration of line 13, so that the following holds at the termination of the `compression_phase()`:
$$\mathcal{T}' = \bigcup_{i=1}^{\max_i} \mathcal{X}_{\mathcal{T}}^{(i)}.$$
Also recall that, by Lemma 3, the `while-loop` at line 2 can be iterated at most $d_{S,T}$ times, so that $\max_i \leq d_{S,T}$. Therefore, when `compression_phase()` terminates, we have $|\mathcal{T}'| \leq |\mathcal{F}|\, d_{S,T}$.

2. *Proof of (2).* In order to prove (2), we exhibit a number of invariants which hold for each iteration of the `while-loop` at line 2 of `compression_phase()`. In what follows, we assume that the procedure `compression_phase()` gets invoked on input $\langle \mathcal{S}, \mathcal{T}, \mathcal{F}, \ell_\uparrow, \ell_\downarrow, d_{S,T}, n \rangle$, and that $\mathcal{S}^{(0)} = \mathcal{S}$ holds by notational convention.

Lemma 6. *Let $i \in [\max_i]$ be any iteration of the `while-loop` at line 2 of `compression_phase()`. Let `p` be any directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}^{(i-1)}$ to $\mathcal{T}$. Then `p` goes either from $\mathcal{X}_\mathcal{S}^{(i)}$ to $\mathcal{T}$ or from $\mathcal{S}^{(i-1)} \setminus \mathcal{X}_\mathcal{S}^{(i)}$ to $\mathcal{X}_\mathcal{T}^{(i)}$. In other words, there exists no directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}^{(i-1)} \setminus \mathcal{X}_\mathcal{S}^{(i)}$ to $\mathcal{T} \setminus \mathcal{X}_\mathcal{T}^{(i)}$.*

Proof. Recall that $\mathcal{X}^{(i)}$ is a vertex cover of the bipartite graph defined as $\mathcal{G}^{(i)} = ((\mathcal{S}^{(i-1)}, \mathcal{T}), \subset)$, which is constructed during the i-th iteration of line 3 within the procedure `compression_phase()`. Also, $\mathcal{X}_\mathcal{S}^{(i)} = \mathcal{X}^{(i)} \cap \mathcal{S}^{(i-1)}$ and $\mathcal{X}_\mathcal{T}^{(i)} = \mathcal{X}^{(i)} \cap \mathcal{T}$, so that the existence of any directed path in $\mathcal{H}_n$ going from $\mathcal{S}^{(i-1)} \setminus \mathcal{X}_\mathcal{S}^{(i)}$ to $\mathcal{T} \setminus \mathcal{X}_\mathcal{T}^{(i)}$ would imply the existence of some edge of $\mathcal{G}^{(i)}$ that would be uncovered by $\mathcal{X}^{(i)}$, contradicting the fact that $\mathcal{X}^{(i)}$ is vertex cover of $\mathcal{G}^{(i)}$. □ □

Lemma 7. *Let $i \in [\max_i]$ be any iteration of the `while-loop` at line 2 of `compression_phase()`. Let U be any subset of $\mathcal{S}^{(i-1)}$ and let V be any subset of $\mathcal{T}$. Let `p` be any directed path in $\mathcal{H}_n$ that goes from U to V. Then `p` goes from $\mathcal{S}$ to V in $\mathcal{H}_n$.*

Proof. Induction on $i \in [\max_i]$.

- *Base Case.* If $i = 1$, recall that $\mathcal{S}^{(0)} = \mathcal{S}$. Then $U \subseteq \mathcal{S}$, which implies the base case.
- *Inductive Step.* Let us assume, by induction hypothesis, that the claim holds for some $i \in [\max_i - 1]$ and let us prove it for $i + 1$. So, let $U \subseteq \mathcal{S}^{(i)}$, and let `p` by any directed path in $\mathcal{H}_n$ that goes from U to V. Recall that $\mathcal{S}^{(i)}$ is the frontier set that is returned by an invocation of `double-bfs_phase()` on input $\mathcal{X}_\mathcal{S}^{(i)}$, at the i-th iteration of line 14, within `compression_phase()`. This amounts to saying that all vertices in $\mathcal{S}^{(i)}$ have been discovered by a BFS starting from $\mathcal{X}_\mathcal{S}^{(i)}$. Recall that $\mathcal{X}_\mathcal{S}^{(i)} = \mathcal{X}^{(i)} \cap \mathcal{S}^{(i-1)}$ so that $\mathcal{X}_\mathcal{S}^{(i)} \subseteq \mathcal{S}^{(i-1)}$. Therefore, `p` is indeed a directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}^{(i-1)}$ to V in $\mathcal{H}_n$. By induction hypothesis, the thesis follows. □

□

Lemma 8. *Let $i \in [\max_i]$ be any index of iteration of the `while-loop` at line 2 of `compression_phase()`. Let `p` be a directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}$ to $\mathcal{T}$ avoiding $\mathcal{F}$. Then, `p` goes either from $\mathcal{X}_\mathcal{S}^{(i)}$ to $\mathcal{T}$ or from $\mathcal{S}$ to $\bigcup_{j=1}^{i} \mathcal{X}_\mathcal{T}^{(j)}$.*

Proof. Induction on $i \in [\max_i]$.

- *Base Case.* If $i = 1$, recall that $\mathcal{S}^{(0)} = \mathcal{S}$. Then, by Lemma 6, we have that p either goes from $X_{\mathcal{S}}^{(1)}$ to $\mathcal{T}$ or from $\mathcal{S} \setminus \mathcal{X}_{\mathcal{S}}^{(1)}$ to $\mathcal{X}_{\mathcal{T}}^{(1)}$. If p goes from $\mathcal{S} \setminus \mathcal{X}_{\mathcal{S}}^{(1)}$ to $\mathcal{X}_{\mathcal{T}}^{(1)}$, then clearly p goes from $\mathcal{S}$ to $\mathcal{X}_{\mathcal{T}}^{(1)}$. This implies the base case.
- *Inductive Step.* Let us assume, by induction hypothesis, that the claim holds for some $i \in [\max_i - 1]$, and let us prove it for $i + 1$. By induction hypothesis, p either goes from $\mathcal{X}_{\mathcal{S}}^{(i)}$ to $\mathcal{T}$ or from $\mathcal{S}$ to $\bigcup_{j=1}^{i} \mathcal{X}_{\mathcal{T}}^{(j)}$ in $\mathcal{H}_n$.
  If p goes from $\mathcal{X}_{\mathcal{S}}^{(i)}$ to $\mathcal{T}$ avoiding $\mathcal{F}$ in $\mathcal{H}_n$, then p must go from $\mathcal{S}^{(i)}$ to $\mathcal{T}$: in fact, recall that $\mathcal{S}^{(i)}$ is the frontier set that is returned by the invocation of `double-bfs_phase()` on input $\mathcal{X}_{\mathcal{S}}^{(i)}$, at line 14 of the `compression_phase()`.
  If p goes from $\mathcal{S}^{(i)}$ to $\mathcal{T}$ then, by Lemma 6, we also have that p goes either from $\mathcal{X}_{\mathcal{S}}^{(i+1)}$ to $\mathcal{T}$ or from $\mathcal{S}^{(i)} \setminus \mathcal{X}_{\mathcal{S}}^{(i+1)}$ to $\mathcal{X}_{\mathcal{T}}^{(i+1)}$ in $\mathcal{H}_n$.
  If p goes from $\mathcal{S}^{(i)} \setminus \mathcal{X}_{\mathcal{S}}^{(i+1)}$ to $\mathcal{X}_{\mathcal{T}}^{(i+1)}$, then p goes from $\mathcal{S}$ to $\mathcal{X}_{\mathcal{T}}^{(i+1)}$ by Lemma 7.
  Since p either goes from $\mathcal{X}_{\mathcal{S}}^{(i+1)}$ to $\mathcal{T}$, or from $\mathcal{S}$ to $\mathcal{X}_{\mathcal{T}}^{(i+1)}$, or from $\mathcal{S}$ to $\bigcup_{j=1}^{i} \mathcal{X}_{\mathcal{T}}^{(j)}$ in $\mathcal{H}_n$, we have that p either goes from $\mathcal{X}_{\mathcal{S}}^{(i+1)}$ to $\mathcal{T}$, or from $\mathcal{S}$ to $\bigcup_{j=1}^{i+1} \mathcal{X}_{\mathcal{T}}^{(j)}$ in $\mathcal{H}_n$, thus concluding the induction and the proof of Lemma 8. □

□

We now have everything we need to prove (2). Let $i = \max_i$ be the last iteration of the `while-loop` at line 2 of `compression_phase()`. Moreover, assume that p is a directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}$ to $\mathcal{T}$ avoiding $\mathcal{F}$.

By Lemma 8, p either goes from $\mathcal{X}_{\mathcal{S}}^{(\max_i)}$ to $\mathcal{T}$ or from $\mathcal{S}$ to $\bigcup_{i=1}^{\max_i} \mathcal{X}_{\mathcal{T}}^{(i)}$. We argue that p cannot go from $\mathcal{X}_{\mathcal{S}}^{(\max_i)}$ to $\mathcal{T}$ in $\mathcal{H}_n$. In fact, any such path must first visit $\mathcal{S}^{(\max_i)}$ in order to reach $\mathcal{T}$. Then, it is sufficient to show that there exists no path that goes from $\mathcal{S}^{(\max_i)}$ to $\mathcal{T}$. Recall that $\max_i$ is the last iteration of the `while-loop` at line 2, and by hypothesis the `compression_phase()` halts by returning $\mathcal{T}'$ at line 16. Therefore, at line 15, it must hold that $\mathcal{S}^{(\max_i)} = \emptyset$ or that both $\ell_{\downarrow}^{(\max_i)} + \ell_{\uparrow} = d_{S,T}$ and $\mathcal{S}^{(\max_i)} \cap \mathcal{T} = \emptyset$. Thus, there exists no directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}^{(\max_i)}$ to $\mathcal{T}$.

Since p does not go from $\mathcal{X}_{\mathcal{S}}^{(\max_i)}$ to $\mathcal{T}$, it must go from $\mathcal{S}$ to $\bigcup_{i=1}^{\max_i} \mathcal{X}_{\mathcal{T}}^{(i)}$ instead; and since $\mathcal{T}' = \bigcup_{i=1}^{\max_i} \mathcal{X}_{\mathcal{T}}^{(i)}$, p must therefore go from $\mathcal{S}$ to $\mathcal{T}'$, which concludes the proof of (2). □

□

Now that we have established the correctness of the procedures it uses, we go back to establishing the correctness of `solve_`Hy-stCon`()`.

Lemma 9. *Each iteration of the* `while-loop` *at line 4 of* `solve_Hy-stCon()` *increases* $\ell_\uparrow + \ell_\downarrow$ *by at least one unit; until* $\ell_\uparrow + \ell_\downarrow = d_{S,T}$ *or until the procedure halts by reaching either line 7, line 10 or line 12.*

Proof. Induction on the index i of iteration of the `while-loop` at line 4.

- *Base Case.* Consider the first iteration of the `while-loop` at line 4. We have $\mathcal{S} = \{S\}$, $\mathcal{T} = \{T\}$, and $\ell_\uparrow = \ell_\downarrow = 0$. Therefore, if $d_{S,T} = 0$, then the procedure halts immediately, either at line 7 (if $S \neq T$) or at line 10 (if $S = T$). If $d_{S,T} > 0$, then a first execution of `double-bfs_phase()` is invoked at line 5, which halts after a finite number of steps by Lemma 1. Notice that the condition for entering the `while-loop` at line 1 of `bfs_phase()` is satisfied, so $\ell_\uparrow + \ell_\downarrow$ gets incremented at line 3 of `bfs_phase()`.

- *Inductive Step.* Assume that at the i-th iteration of the `while-loop` at line 4, we have $\ell_\uparrow + \ell_\downarrow < d_{S,T}$. Furthermore, assume that none of the conditions checked by `solve_Hy-stCon()` at line 6, line 8 and line 12 are satisfied. Then, the procedure does not halt at i-th iteration. Recall that `double-bfs_phase()`, which is invoked at line 5, halts within finite time by Lemma 1; also, recall that `compression_phase()`, which is invoked at line 11, halts within finite time by Lemma 3. Thus, at the end of the i-th iteration, line 13 gets finally executed. At line 13, the current frontier $\mathcal{T}$ gets replaced by the value $\mathcal{T}'$, previously returned by `compression_phase()` at line 11. Notice that $|\mathcal{T}'| \leq |\mathcal{F}|\, d_{S,T}$ holds by Proposition 4. The $(i+1)$-th iteration of the `while-loop` at line 4 starts at this point. Then, at line 5, another round of `double-bfs_phase()` is executed. If $\mathcal{T} \neq \emptyset$ and $\ell_\uparrow + \ell_\downarrow < d_{S,T}$, the condition for entering the `while-loop` at line 1 of `bfs_phase()` is satisfied, so that $\ell_\uparrow + \ell_\downarrow$ gets incremented at line 3. If $\mathcal{T} = \emptyset$ or $\ell_\uparrow + \ell_\downarrow = d_{S,T}$, then the procedure halts at line 7. This implies that the invariant is maintained for each iteration i. □

□

Proposition 5. *The procedure* `solve_Hy-stCon()` *halts within a finite number of steps. In particular, the* `while-loop` *at line 4 iterates at most* $d_{S,T}$ *times.*

Proof. Recall the statement of Lemma 9. As soon as $\ell_\uparrow + \ell_\downarrow = d_{S,T}$, then `solve_Hy-stCon()` halts either at line 7 (if $\mathcal{S} \cap \mathcal{T} = \emptyset$) or at line 10 (if $\mathcal{S} \cap \mathcal{T} \neq \emptyset$). In particular, this implies that the `while-loop` at line 4 of the `solve_Hy-stCon()` can be iterated at most $d_{S,T}$ times. □ □

Proposition 6. *Assume that* `solve_Hy-stCon()` *halts by returning the pair* $\langle$`YES`, p$\rangle$. *Then* p *is a directed path in* $\mathcal{H}_n$ *that goes from* S *to* T *avoiding* $\mathcal{F}$.

Proof. Observe that `solve_Hy-stCon()` can return $\langle$`YES`, p$\rangle$ as output only at line 10 or at line 12. In the latter case, p gets constructed at line 11 by invoking `compression_phase()`, so the thesis follows by Lemma 5. Otherwise, assume that p is returned at line 10. Therefore, at the last iteration of line 8, it must hold that $\mathcal{S} \cap \mathcal{T} \neq \emptyset$. Then, let $X \in \mathcal{S} \cap \mathcal{T}$. By Lemma 4 there exists a directed path $p_{S,X}$ in $\mathcal{H}_n$ that goes from S to X avoiding $\mathcal{F}$ (because $X \in \mathcal{S}$), and there exists another directed path $\mathsf{p}_{X,T}$ in $\mathcal{H}_n$ that goes from X to T avoiding $\mathcal{F}$ (because

$X \in \mathcal{T}$). Therefore, `reconstruct_path()` at line 9, is able to compute a directed path $\mathtt{p} = \mathtt{p}_{S,X}\mathtt{p}_{X,T}$ in $\mathcal{H}_n$ that goes from S to T avoiding $\mathcal{F}$, which gets returned at line 12. □ □

Lemma 10. *The following invariant is maintained at each line of* `solve_Hy-stCon()`*. If* p *is any directed path in $\mathcal{H}_n$ that goes from S to T avoiding $\mathcal{F}$, then* p *goes from $\mathcal{S}$ to $\mathcal{T}$.*

Proof. Induction on the index i of iteration of the `while-loop` at line 2.

- *Base Case.* Before entering the first iteration, since $\mathcal{S} = \{S\}$ and $\mathcal{T} = \{T\}$, the thesis holds.
- *Inductive Step.* Assume that the thesis holds at the end of the i-th iteration. So, let $\mathcal{S}^{(i)}$ and $\mathcal{T}^{(i)}$ be the frontier sets at the end of the i-th iteration. When $i = 0$, just recall that $\mathcal{S}^{(0)} = \{S\}$ and $\mathcal{T}^{(0)} = \{T\}$. Now, at the beginning of the $(i+1)$-th iteration, in particular at line 5 of `solve_Hy-stCon()`, let $\mathcal{S}$ and $\mathcal{T}$ be the frontier sets returned by the invocation of `double-bfs_phase()`. If p is any directed path in $\mathcal{H}_n$ that goes from S to T avoiding $\mathcal{F}$, then p goes from $\mathcal{S}^{(i)}$ to $\mathcal{T}^{(i)}$ by induction hypothesis. It is not difficult to see that if p goes from $\mathcal{S}^{(i)}$ to $\mathcal{T}^{(i)}$ avoiding $\mathcal{F}$, then p must go from $\mathcal{S}$ to $\mathcal{T}$ as well: at this point, the reader can check that this is a direct consequence of `double-bfs_phase()`'s construction. If the $(i+1)$-th iteration doesn't halt, then the `compression_phase()` at line 11 gets invoked. Then, let $\mathcal{T}'$ be the value returned by `compression_phase()` at line 11. By Proposition 4, if p is a directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}$ to $\mathcal{T}$ avoiding $\mathcal{F}$, then p goes from $\mathcal{S}$ to $\mathcal{T}'$. Thus, it is indeed correct to update $\mathcal{T}$ by $\mathcal{T}'$ at line 13 of `solve_Hy-stCon()`. This implies that the thesis holds for each iteration of the `while-loop` at line 2, until termination. □

□

Proposition 7. *Assume that* `solve_Hy-stCon()` *halts by returning* `NO`*. Then there is no directed path in $\mathcal{H}_n$ that goes from S to T avoiding $\mathcal{F}$.*

Proof. Since `solve_Hy-stCon()` returns `NO`, the condition checked at line 6 must be satisfied: if $\mathcal{S} = \emptyset$ or $\mathcal{T} = \emptyset$, then there exists no directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}$ to $\mathcal{T}$; similarly, if $\ell_\uparrow + \ell_\downarrow = d_{S,T}$ and $\mathcal{S} \cap \mathcal{T} = \emptyset$, then there exists no directed path in $\mathcal{H}_n$ that goes from $\mathcal{S}$ to $\mathcal{T}$. By Lemma 10, there exists no directed path in $\mathcal{H}_n$ that goes from S to T avoiding $\mathcal{F}$. □ □

Theorem 5 follows, at this point, from Propositions 5 to 7.

B.3 Complexity Analysis

We now analyze the time complexity of `solve_Hy-stCon()`, starting with that of the procedures it relies on.

Lemma 11. *The* `double-bfs_phase()` *always halts within $O(|\mathcal{F}|\, d_{S,T}^2\, n)$ time.*

Proof. It is sufficient to prove that `bfs_phase()` always halts within $O(|\mathcal{F}|\, d_{S,T}^2\, n)$ time. Recall that, by Lemma 1, the `while-loop` at line 1 of `bfs_phase()` iterates at most $d_{S,T}$ times. At each iteration, `next_step_bfs()` gets invoked on some input set $\mathcal{X} \in \wp_n$ and flag variable $\texttt{drt} \in \{\texttt{in}, \texttt{out}\}$ (see line 2 of `bfs_phase()`).

We argue that each of these invocations takes at most $O(|\mathcal{F}|\, d_{S,T}\, n)$ time. Assume that $N^{\texttt{drt}}$ is $N^{\texttt{in}}$ when $\texttt{drt} = \texttt{in}$, and that it is $N^{\texttt{out}}$ otherwise. Then, each invocation of `next_step_bfs()` takes $O(|\mathcal{X}| \max_{v \in \mathcal{X}}\{|N^{\texttt{drt}}(v)|\})$ time, because it involves visiting $N^{\texttt{drt}}(v)$ for each $v \in \mathcal{X}$; still, in order to enter the `while-loop` at line 1 of `bfs_phase()`, we must have $|\mathcal{X}| \leq |\mathcal{F}|\, d_{S,T}$, and moreover we have $|N^{\texttt{drt}}(v)| = O(n)$ for every $v \in \mathcal{X}$. Since the total number of iterations is bounded above by $d_{S,T}$, the bound follows. □ □

Lemma 12. *Assume that `compression_phase()` gets invoked at line 11 of the procedure `solve_`*Hy-stCon*`()`. If `compression_phase()` halts without ever executing the procedure `compute_Lehman-Ron_paths()` at line 8, then it halts within the following time bound:*

$$O\left(\min\left(\sqrt{|\mathcal{F}|\, d_{S,T}\, n}, |\mathcal{F}|\right) |\mathcal{F}|^2\, d_{S,T}^3\, n^2\right). \tag{1}$$

Otherwise, if `compression_phase()` executes `compute_Lehman-Ron_paths()` at line 8, then it halts within the following time bound:

$$O\left(\min\left(\sqrt{|\mathcal{F}|\, d_{S,T}\, n}, |\mathcal{F}|\right) |\mathcal{F}|^2\, d_{S,T}^3\, n^2 + |\mathcal{F}|^{5/2} n^{3/2} d_{S,T}\right). \tag{2}$$

Proof. We start with some preliminary observations that will be useful in proving time bounds (1) and (2). Let us assume that `compression_phase()` is invoked on the following input $\langle \mathcal{S}, \mathcal{T}, \mathcal{F}, \ell_\uparrow, \ell_\downarrow, d_{S,T}, n\rangle$ at line 11 of `solve_`Hy-stCon`()`. We argue that the following bounds hold on the size of $\mathcal{S}$ and $\mathcal{T}$:

$$|\mathcal{S}| \leq |\mathcal{F}|\, d_{S,T}\, n \quad \text{and} \quad |\mathcal{T}| \leq |\mathcal{F}|\, d_{S,T}\, n. \tag{3}$$

In fact, notice that $\mathcal{S}$ and $\mathcal{T}$ were computed during a previous invocation of `double-bfs_phase()`, at line 5 of `solve_`Hy-stCon`()`. Therefore, it suffices to consider the set $\mathcal{X}$ which is computed by passing through the `while-loop` at line 1 of `bfs_phase()`. The condition for entering that `while-loop` requires $|\mathcal{X}| \leq |\mathcal{F}|\, d_{S,T}$. Therefore, as soon as `bfs_phase()` exits that `while-loop`, we must have $|\mathcal{X}| \leq |\mathcal{F}|\, d_{S,T}\, n$. This implies the bounds specified by (3).

Let us now consider the bipartite graph $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G}) = ((\mathcal{S}, \mathcal{T}), \subset)$, which is constructed at line 3 of `compression_phase()`. Since

$$|V_\mathcal{G}| = |\mathcal{S}| + |\mathcal{T}| \leq 2\, |\mathcal{F}|\, d_{S,T}\, n,$$

we also have the following bound on the size of its edge set:

$$|E_\mathcal{G}| \leq |V_\mathcal{G}|^2 \leq 4\, |\mathcal{F}|^2\, d_{S,T}^2\, n^2.$$

We can now proceed with the proof of the two time bounds.

1. In the case where `compute_Lehman-Ron_paths()` never gets executed, recall that, at line 4, the `compression_phase()` computes a matching $\mathcal{M}$ of $\mathcal{G}$ such that $|\mathcal{M}| = \min(m^*, |\mathcal{F}| + 1)$, where m^* is the size of a maximum cardinality matching of $\mathcal{G}$. The Hopcroft-Karp algorithm [11] performs that task within the following time bound $t_{\mathcal{M}}$:

$$t_{\mathcal{M}} = O\left(\min(\sqrt{|V_{\mathcal{G}}|}, |\mathcal{F}|)\,|E_{\mathcal{G}}|\right) = O\left(\min\left(\sqrt{|\mathcal{F}|\,d_{S,T}\,n}, |\mathcal{F}|\right)|\mathcal{F}|^2\,d_{S,T}^2\,n^2\right).$$

   Notice that the time complexity of `compute_min_vertex_cover()`, which is invoked at line 11 of `compression_phase()`, is bounded above by the time complexity of computing $\mathcal{M}$ at line 4. Also, by Lemma 11, the time complexity of the `double-bfs_phase()`, which is invoked at line 14 of `compression_phase()`, is bounded above by the same quantity.

   If `compute_Lehman-Ron_paths()` never gets executed at line 8, then during each iteration of the `while-loop` at line 2 of `compression_phase()`, the most expensive task is that of computing the matching $\mathcal{M}$ at line 4. Recall that, according to Lemma 3, the `while-loop` at line 2 iterates at most $d_{S,T}$ times. We conclude that, in this case, the `compression_phase()` halts within the following time bound:

$$t_{\mathcal{M}}\,d_{S,T} = O\left(\min\left(\sqrt{|\mathcal{F}|\,d_{S,T}\,n}, |\mathcal{F}|\right)|\mathcal{F}|^2\,d_{S,T}^3\,n^2\right).$$

2. In the case where `compute_Lehman-Ron_paths()` gets executed, which happens whenever $|\mathcal{M}| = |\mathcal{F}| + 1$, we must now take its time complexity into account, which we analyze below.

   First, consider the set $\mathcal{M}_{\mathcal{S}}$ computed at line 6 of `compression_phase()`. The following bound holds on its size:

$$|\mathcal{M}_{\mathcal{S}}| = |\mathcal{M}| = |\mathcal{F}| + 1.$$

   The same bound holds for the set $\mathcal{M}_{\mathcal{T}} \subseteq \mathcal{T}$ which is computed at line 7 — namely: $|\mathcal{M}_{\mathcal{T}}| = |\mathcal{M}| = |\mathcal{F}| + 1$. By Theorem 3, provided that we consider the parameter $m = |\mathcal{M}| = O(|\mathcal{F}|)$, invoking `compute_Lehman-Ron_paths()` on input $\langle \mathcal{M}_{\mathcal{S}}, \mathcal{M}_{\mathcal{T}}, \mathcal{M}, n\rangle$ takes time at most t_{LR}, where:

$$t_{\mathrm{LR}} = O\left(m^{5/2} n^{3/2} d_{S,T}\right) = O\left(|\mathcal{F}|^{5/2} n^{3/2} d_{S,T}\right).$$

   Recall that, by Lemma 3, the `while-loop` at line 2 iterates at most $d_{S,T}$ times. At each of such iterations, a brand new matching $\mathcal{M}$ gets computed at line 4. Finally, at the very last of such iterations, provided that $|\mathcal{M}| > |\mathcal{F}|$, then the procedure `compute_Lehman-Ron_paths()` is invoked at line 8. Therefore, we conclude that whenever `compression_phase()` executes `compute_Lehman-Ron_paths()` at line 8, then it halts within the following time bound:

$$t_{\mathcal{M}}\,d_{S,T} + t_{\mathrm{LR}} = O\left(\min\left(\sqrt{|\mathcal{F}|\,d_{S,T}\,n}, |\mathcal{F}|\right)|\mathcal{F}|^2\,d_{S,T}^3\,n^2 + |\mathcal{F}|^{5/2} n^{3/2} d_{S,T}\right).$$

□

□

Proposition 8. *The* Decision-Task *of* Hy-stCon *can be solved within the following time bound on any input* $\langle S, T, \mathcal{F}, n\rangle$*:*

$$O\left(\min\left(\sqrt{|\mathcal{F}|\, d_{S,T}\, n}, |\mathcal{F}|\right)|\mathcal{F}|^2\, d_{S,T}^4\, n^2\right).$$

Proof. Let us consider the procedure `solve_Hy-stCon()` of Algorithm 1. By Proposition 5, the `while-loop` at line 4 iterates at most $d_{S,T}$ times. At each iteration, `double-bfs_phase()` is invoked at line 5, and `compression_phase()` is invoked soon after at line 11. By Lemma 11, the most expensive one between the two procedures is clearly `compression_phase()`. Recall that, if we are content with solving the Decision-Task of Hy-stCon, then the `compression_phase()` can be implemented so that it always halts without ever executing the procedure `compute_Lehman-Ron_paths()` at line 8. Therefore, by Lemma 12, each invocation of `compression_phase()` takes time at most

$$O\left(\min\left(\sqrt{|\mathcal{F}|\, d_{S,T}\, n}, |\mathcal{F}|\right)|\mathcal{F}|^2\, d_{S,T}^3\, n^2\right).$$

Since we have at most $d_{S,T}$ of such invocations, then the thesis follows. □ □

Proposition 9. *The* Search-Task *of* Hy-stCon *can be solved within the following time bound on any input* $\langle S, T, \mathcal{F}, n\rangle$*:*

$$O\left(\min\left(\sqrt{|\mathcal{F}|\, d_{S,T}\, n}, |\mathcal{F}|\right)|\mathcal{F}|^2\, d_{S,T}^4\, n^2 + |\mathcal{F}|^{5/2} n^{3/2} d_{S,T}\right).$$

Proof. Let us consider the procedure `solve_Hy-stCon()` of Algorithm 1. By Proposition 5, the `while-loop` at line 4 iterates at most $d_{S,T}$ times. At each iteration, `double-bfs_phase()` is invoked at line 5, and `compression_phase()` is invoked shortly after at line 11. By Lemma 11, the most expensive step between the two is clearly the `compression_phase()`. Recall that, if we aim to solve the Search-Task of Hy-stCon, then the `compression_phase()` possibly executes the `compute_Lehman-Ron_paths()` procedure at line 8. Nevertheless, whenever `compression_phase()` executes `compute_Lehman-Ron_paths()` at line 8, then the procedure `solve_Hy-stCon()` halts shortly after at line 12. This means that the only invocation of `compression_phase()` that possibly executes `compute_Lehman-Ron_paths()` is the very last invocation. Then, each invocation of `compression_phase()`, except the very last one, halts within the following time bound by Lemma 12:

$$O\big(\min\big(\sqrt{|\mathcal{F}|\, d_{S,T}\, n}, |\mathcal{F}|\big)|\mathcal{F}|^2\, d_{S,T}^3\, n^2\big).$$

Since the very last invocation of `compression_phase()` possibly executes the procedure `compute_Lehman-Ron_paths()` at line 8, the following time bound holds on the last invocation of `compression_phase()` by Lemma 12:

$$O\left(\min\left(\sqrt{|\mathcal{F}|\, d_{S,T}\, n}, |\mathcal{F}|\right)|\mathcal{F}|^2\, d_{S,T}^3\, n^2 + |\mathcal{F}|^{5/2} n^{3/2} d_{S,T}\right).$$

Since there are at most $d_{S,T}$ invocations of the `compression_phase()`, the thesis follows. □ □