Prefix Reversals on Binary and Ternary Strings*

Cor Hurkens[1], Leo van Iersel[1], Judith Keijsper[1], Steven Kelk[2], Leen Stougie[1,2], and John Tromp[2]

[1] Technische Universiteit Eindhoven (TU/e), Den Dolech 2, 5612 AX Eindhoven, Netherlands
wscor@win.tue.nl, l.j.j.v.iersel@tue.nl, j.c.m.keijsper@tue.nl
[2] Centrum voor Wiskunde en Informatica (CWI), Kruislaan 413, 1098 SJ Amsterdam, Netherlands
S.M.Kelk@cwi.nl, Leen.Stougie@cwi.nl, John.Tromp@cwi.nl

Abstract. Given a permutation π, the application of prefix reversal $f^{(i)}$ to π reverses the order of the first i elements of π. The problem of Sorting By Prefix Reversals (also known as *pancake flipping*), made famous by Gates and Papadimitriou (Bounds for sorting by prefix reversal, *Discrete Mathematics* 27, pp. 47-57), asks for the minimum number of prefix reversals required to sort the elements of a given permutation. In this paper we study a variant of this problem where the prefix reversals act not on permutations but on strings over a fixed size alphabet. We determine the minimum number of prefix reversals required to sort binary and ternary strings, with polynomial-time algorithms for these sorting problems as a result; demonstrate that computing the minimum prefix reversal distance between two binary strings is NP-hard; give an exact expression for the prefix reversal diameter of binary strings, and give bounds on the prefix reversal diameter of ternary strings. We also consider a weaker form of sorting called *grouping* (of identical symbols) and give polynomial-time algorithms for optimally grouping binary and ternary strings. A number of intriguing open problems are also discussed.

1 Introduction

For a permutation $\pi = \pi(0)\pi(1)\dots\pi(n-1)$ the application of *prefix reversal* $f^{(i)}$, which we call *flip* for short, to π reverses the order of the first i elements: $f^{(i)}(\pi) = \pi(i-1)\dots\pi(0)\pi(i)\dots\pi(n-1)$. The problem of *Sorting By Prefix Reversals* (MIN-SBPR), popularised by Gates and Papadimitriou [11] and often referred to as the *pancake flipping problem*, is defined as follows: given a permutation π of $\{0, 1, \dots, n-1\}$, determine its sorting distance i.e. the smallest number of flips required to transform π into the identity permutation $01\dots(n-1)$.[1]

MIN-SBPR arises in the context of computational biology when seeking to explain the genetic difference between two given species by the most parsimonious (i.e. shortest) sequence of gene rearrangements. It is one of a family of

* This research has been funded by the Dutch BSIK/BRICKS project.

[1] We adopt the convention of numbering from 0 rather than 1.

H. Anai, K. Horimoto, and T. Kutsia (Eds.): AB 2007, LNCS 4545, pp. 292–306, 2007.

genome rearrangement operations that also includes arbitrary (substring) reversals [13], *transpositions* (where two adjacent substrings are swapped) [6], and *translocations* (whereby, in the context of multiple chromosomes, the exchange of chromosome-ends is simulated) [2]. To extend biological relevance it is now commonplace that these operations are studied not only in isolation but also in (weighted) combination with each other [1]; MIN-SBPR is arguably most biologically applicable in the context of such weighted combinations. MIN-SBPR also has relevance in the area of efficient network design [14,16].

The computational complexity of MIN-SBPR, however, remains open. A recent 2-approximation algorithm [8] is currently the best-known approximation result[2]. Indeed, most studies to date have focused not on the computational complexity of MIN-SBPR but rather on determining the worst-case sorting distance $wc(n)$ over all length-n permutations i.e. the "worst case scenario" for length-n permutations. From [11] and [14] we know that $(15/14)n \leq wc(n) \leq (5n+5)/3$.

A natural variant of MIN-SBPR is to consider the action of flips not on permutations but on strings over fixed size alphabets. This shift is inspired by the biological observation that multiple "copies" of the same gene can appear at various places along the genome, although this does not lead to the bounded size alphabets that we will study here. The shift from permutations to strings alters the problem universe somewhat. With permutations, for example, the *distance problem*, i.e. given two permutations π_1 and π_2, determine the smallest number of flips required to transform π_1 into π_2, is equivalent to sorting, because the symbols can simply be relabelled to make either permutation equal to the identity permutation. For strings like 101, such a relabelling is not possible. Thus, the distance problem on string pairs appears to be strictly more general than the sorting problem on strings, naturally defined as putting all elements in non-descending order.

Indeed, papers by Christie and Irving [4] and Radcliffe, Scott and Wilmer [17] explore the consequences of switching from permutations to strings; they both consider arbitrary (substring) reversals and transpositions. It has been noted that, viewed as a whole, such rearrangement operations on strings have bearing on the study of orthologous gene assignment [3], especially where the level of symbol repetition in the strings is low. There is also a somewhat surprising link with the relatively unexplored family of *string partitioning* problems [12]. To put our work in context, we briefly describe the most relevant (for this paper) results from [4] and [17].

The earlier paper [4], gives, in both the case of reversals and transpositions, polynomial-time algorithms for computing the minimum number of operations to sort a given binary string, as well as exact, constructive diameter results on binary strings. Additionally, their proof that computing the reversal distance between strings is NP-hard, supports the intuition that distance problems are harder than sorting problems on strings. They present upper and lower bounds for computing reversal and transposition distance on binary strings.

[2] Although not explicitly described as such, the algorithm provided ten years earlier in [5] is a 2-approximation algorithm for the *signed* version of the problem.

The more recent paper [17] gives refined and generalised reversal diameter results for non-fixed size alphabets. It also gives a polynomial-time algorithm for optimally sorting a ternary (3 letter alphabet) string with reversals. The authors refer to the prefix reversal counterparts of these (and other) results as interesting open problems. They further provide an alternative proof of Christie and Irving's NP-hardness result for reversals, and sketch a proof that computing the *transposition distance* between binary strings is NP-hard. As we later note, this proof can also be used to obtain a specific reducibility result for prefix reversals. They also have some first results on approximation (giving a PTAS - a *Polynomial-Time Approximation Scheme* - for computing the distance between *dense instances*) and on the distance between random strings, both of which apply to prefix reversals as well.

In this paper we supplement results of [4] and [17] by their counterparts on prefix reversals. In Section 3 (*Grouping*) we introduce a weaker form of sorting where identical symbols need only be grouped together, while the groups can be in any order. For grouping on binary and ternary strings we give a complete characterisation of the minimum number of flips required to group a string, and provide polynomial-time algorithms for computing such an optimal sequence of flips. (The complexity of grouping over larger fixed size alphabets remains open but as an intermediate result we describe how a PTAS can be constructed for each such problem.) Grouping aids in developing a deeper understanding of sorting which is why we tackle it first. It was also mentioned as a problem of interest in its own right by Eriksson et al. [7]. Then, in Section 4 (*Sorting*), we give polynomial-time algorithms (again based on a complete characterisation) for optimally sorting binary and ternary strings with flips. (The complexity of sorting also remains open for larger fixed size alphabets. As with grouping we thus provide, as an intermediate result, a PTAS for each such problem.) In Section 5 we show that the flip diameter (i.e. the maximum distance between any two strings) on binary strings is $n-1$, and on ternary strings (for $n > 3$) lies somewhere between $n-1$ and $(4/3)n$, with empirical support for the former. In Section 6 we show that the flip distance problem on binary strings is NP-hard, and point out that a reduction in [17] also applies to prefix reversals, showing that the flip distance problem on *arbitrary* strings is polynomial-time reducible (in an approximation-preserving sense) to the binary problem. We conclude in Section 7 with a discussion of some of the intriguing open problems that have emerged during this work. Indeed, our initial exploration has identified many basic (yet surprisingly difficult) combinatorial problems that deserve further analysis.

2 Preliminaries

Let $[k]$ denote the first k non-negative integers $\{0, 1, ..., k-1\}$. A k-ary string is a string over the alphabet $[k]$, while a string s is said to be *fully* k-ary, or to *have arity* k, if the set of symbols occuring in it is $[k]$.

We index the symbols in a string s of length n from 1 through n: $s = s_1 s_2 \dots s_n$. Two strings are *compatible* if they have the same symbol frequencies

(and hence the same length), e.g. 0012 and 1002 are compatible but 0012 and 0112 are not. For a given string s, let $I(s)$ be the string obtained by sorting the symbols of s in non-descending order e.g. $I(1022011) = 0011122$. The prefix reversal (flip for short) $f^{(i)}(s)$ reverses the length i prefix of its argument, which should have length at least i. Alternatively, we denote application of $f^{(i)}(s)$ by underlining the length i prefix. Thus, $f^{(2)}(2012) = \underline{20}12 = 0212$ and $f^{(3)}(2012) = \underline{201}2 = 1022$. The *flip distance* $d(s, s')$ between two strings s and s' is defined as the smallest number of flips required to transform s into s', if they are compatible and ∞ otherwise. Since a flip is its own inverse, flip distance is symmetric.

The *flip sorting distance* $d_{\mathrm{s}}(s) = d(s, I(s))$ of a string s is defined as the number of flips of an *optimal sorting sequence* to transform s into $I(s)$. An algorithm sorts s optimally if it computes an optimal sorting sequence for s.

In the next two sections we consider strings to be equivalent if one can be transformed into the other by repeatedly duplicating symbols and eliminating one of two adjacent identical symbols. As representatives of the equivalence classes we take the shortest string in each class. These are exactly the strings in which adjacent symbols always differ. We express all flip operations in terms of these *normalized* strings. E.g. we write $f^{(3)}(2012) = \underline{201}2 = 102$. A flip that brings two identical symbols together, thereby shortening the string by 1, is called a *1-flip*, while all others, that leave the string length invariant, are called *0-flips*.

We follow the standard notation for regular expressions: Superindex ^{i} on a substring denotes the number of repetitions of the substring, with * and $^+$ denoting 0-or-more and 1-or-more repetitions, respectively, ϵ denotes the empty string, brackets of the form $\{\}$ are used to denote that a symbol can be exactly one of the elements within the brackets, and the product sign $\prod$ denotes concatenation of an indexed series. For example $\prod_{i=1}^{3}(10^i2) = 102100210002$, and $\{1, 01\}^*\{\epsilon, 0\}$ denotes the set of binary strings with no 00 substring.

3 Grouping

The task of sorting a string can be broken down into two subproblems: *grouping* identical symbols together and putting the groups of identical symbols in the right *order*. Notice that first grouping and then ordering may not be the most efficient way to sort strings. Although grouping appears to be slightly easier than the sorting problem, essentially the same questions remain open as in sorting. Grouping binary strings is trivial and in Section 3.1 we give the grouping distances of all ternary strings. As a result we give polynomial time algorithms for binary and ternary grouping. For larger alphabets the grouping problem remains open; as an intermediate result we describe in Section 3.2 a PTAS for each such problem. While the problems of grouping and sorting are closely related for strings on small alphabets, the problems diverge when alphabet size approaches the string length, with permutations being the limit.

Recall that we consider only normalized strings, as representatives of equivalence classes. The *flip grouping distance* $d_{\mathrm{g}}(s)$ of a fully k-ary string s is defined as the minimum number of flips required to reduce the string to one of length k.

3.1 Grouping Binary and Ternary Strings

Lemma 1. *$d_{\mathrm{g}}(s) \geq n - k$ for any fully k-ary string s of length n.*

Proof. The proof follows from the observations that, after grouping, fully k-ary string s has length k and that each flip can shorten s by at most 1. □

Lemma 2. *$d_{\mathrm{g}}(s) \leq n - 2$ for any fully k-ary string s of length n.*

Proof. Consider the following simple algorithm. If the leading symbol occurs elsewhere then a 1-flip bringing them together exists, so perform this 1-flip. If not, then we use a 0-flip to put this symbol in front of a suffix in which we accumulate uniquely appearing symbols. Repeat until the string is grouped.

Clearly no more than $n-k$ 1-flips will be necessary. Also, no more than $k-2$ 0-flips will ever be necessary, because after $k-2$ 0-flips the prefix of the string will consist of only two types of symbol, and the algorithm will never perform a 0-move on such a string. Thus at most $(n-k)+(k-2) = n-2$ flips in total will be needed. □

As a corollary we obtain the grouping distance of binary strings.

Theorem 1. *$d_{\mathrm{g}}(s) = n - 2$ for any fully binary string s of length n.* □

We will now define a class of bad ternary strings and prove that these are the only ternary strings that need $n-2$ rather than $n-3$ flips to be grouped.

Definition 1. *We define* bad *strings as all fully ternary strings of one of the following types, up to relabeling:*

I. strings of length greater than 3, in which the leading symbol appears only once: $0(12)^{\geq 2}$ *and* $02(12)^{+}$
II. strings having identical symbols at every other position, starting from the last: $(\{0,1\}2)^{+}$ *and* $(2\{0,1\})^{+}2$
III. odd length strings whose leading symbol appears exactly once more, at an even position, and both occurrences are followed by the same symbol: $0(21)^{+}02(12)^{*}$
IV. the following strings:
$X_1 = 210212$, $X_2 = 021012$, $X_3 = 0120212$, $X_4 = 1201212$, $X_5 = 02101212$, $X_6 = 20210212$, $X_7 = 020210212$, $X_8 = 120120212$.

All other fully ternary strings are good. *Strings of type I, II and III, shortly I-, II-, and III-strings, respectively, are called generically bad, or* g-bad *for short.*

Lemma 3. *$d_{\mathrm{g}}(s) = n - 2$ if ternary string s of length n is bad.*

Proof. Because of Lemmas 1 and 2, it suffices to show that in each case a 0-flip is necessary: I-strings admit only 0-flips. A 1-flip on a II-string leads to a II-string and eventually to a I-string. Any III-string admits only one 1-flip leading to a II-string. For IV-strings it is easy to check that each possible 1-flip leads to either a shorter IV-string, or to a I-,II-, or III-string. A full proof can be found in [15]. □

Lemma 4. *$d_g(s) = n - 3$ if ternary string s of length n is good.*

Proof. The proof is by induction on n. The induction basis for $n = 3$ is trivial. We show the statement for strings of length $n + 1$ by showing that if a bad string s' of length n can be obtained through a 1-flip from a good (*parent*) string s of length $n + 1$, then s admits another 1-flip which leads to a good string. Note that a 1-flip $f^{(i)}(s) = s'$ brings symbols s_1 and s_{i+1} together, hence $s_1 = s_{i+1} \neq s_i = s'_1$ which shows that the symbol deleted from *parent* s differs from the leading symbol of *child* s'. We enumerate all possible bad child strings s' and distinguish cases based on the leading symbol of good parent s.

For IV-strings, Table 1 lists all parents with, for each good parent, a 1-flip to a good string. It remains to prove that for each g-bad string all parents are either bad or have a g-1-flip, defined as a 1-flip resulting in a string that is not g-bad (i.e. either good or of type IV).

Type I, odd: $0(12)^{\geq 2}$ has possible parents starting with:
- 1: $1(21)^i012(12)^j$ with $i + j > 0$:
  - **If** $i > 0$ there is a g-1-flip $\underline{12}1(21)^{i-1}012(12)^j = (21)^i012(12)^j$;
  - **If** $i = 0$ and $j > 0$ there is a g-1-flip $\underline{1012}(12)^j = 210(12)^j$;
- 2: $21(21)^i02(12)^j$ with $i + j > 0$.
  - **If** $i > 0$ there is a g-1-flip $\underline{21}(21)^i02(12)^j = 1(21)^i02(12)^j$;
  - **If** $i = 0$ and $j > 1$ there is a g-1-flip $\underline{210212}(12)^{j-1} = 120(12)^j$;
  - **If** $i = 0$ and $j = 1$ the parent is $210212 = X_1$.

Type I, even: these strings are also of type II, see below.

Type II, odd: $(2\{0,1\})^+2$ has only parents of type II.

Type II, even: $02(\{0,1\}2)^*$ has possible parents starting with:
- 2: $2(\{0,1\}2)^*$ is of type II;
- 1: $12(\{0,1\}2)^*012(\{0,1\}2)^*$ with three cases for a possible third 1:
  - **None:** parent is $12(02)^*012(02)^*$, which is of type III;
  - **Before** 01**:** then there is a g-1-flip
    $\underline{12(\{0,1\}2)^*12(\{0,1\}2)^*012(\{0,1\}2)^*}$
    $= 2(\{0,1\}2)^*12(\{0,1\}2)^*012(\{0,1\}2)^*$;
  - **After** 01**:** then there is a g-1-flip
    $\underline{12(\{0,1\}2)^*012(\{0,1\}2)^*12(\{0,1\}2)^*}$
    $= 2(\{0,1\}2)^*102(\{0,1\}2)^*12(\{0,1\}2)^*$.

Type III: $0(21)^+02(12)^*$ has possible parents starting with:
- 1: $(12)^i01(21)^j02(12)^k$ with $i > 0$:
  - **If** $i > 1$ there is a g-1-flip
    $\underline{12}(12)^{i-1}01(21)^j02(12)^k = 2(12)^{i-1}01(21)^j02(12)^k$;
  - **If** $i = 1, j > 0$ there is a g-1-flip
    $\underline{12012}1(21)^{j-1}02(12)^k = 21021(21)^{j-1}02(12)^k$;

If $i = 1, j = 0, k > 0$ there is a g-1-flip $\underline{120102}(12)^k = 20102(12)^k$;
If $i = 1, j = k = 0$ then the parent is $120102 = X_2$ (relabelled);
1: $(12)^+0(12)^+0(12)^+$: there is a g-1-flip
$\underline{(12)^+0}(12)^+0(12)^+ = 0(21)^+20(12)^+$;
2: $2(12)^*0(21)^+02(12)^*$: there is a g-1-flip
$\underline{2(12)^*0(21)^+02}(12)^* = 0(12)^+0(21)^*2$;
2: $(21)^i20(12)^j02(12)^k$ with $j > 0$:
If $i = 0, j = 1$ then the parent is $210212 = X_1$;
If $i + j > 1$ then $\underline{(21)^i2012}(12)^{j-1}02(12)^k = 102(12)^{i+j-1}02(12)^k$ is a g-1-flip. □

Table 1. Type IV strings, their parents, and for each good parent, a 1-flip to a good string

IV-String	Parents
$X_1 = 210212$	$\underline{1210}212$, $0120212 = X_3$, $1201212 = X_4$
$X_2 = 021012$	$\underline{2021012}$, $\underline{1201}012$, $\underline{10}12012$, $\underline{2010}212$
$X_3 = 0120212$	$\underline{10120}212$, $\underline{21020}212$, $20210212 = X_6$, $\underline{1202}1012$, $\underline{20}212012$
$X_4 = 1201212$	$\underline{21201}212$, $02101212 = X_5$, $\underline{21021212}$, $\underline{21}210212$
$X_5 = 02101212$	$\underline{202101}212$, $\underline{120101}212$, $\underline{10}1201212$, $\underline{2101}20212$, $\underline{12}1012012$, $\underline{20}2010212$
$X_6 = 20210212$	$020210212 = X_7$, $\underline{120210}212$, $\underline{012020}212$, $120120212 = X_8$
$X_7 = 020210212$	$\underline{2020210}212$, $\underline{2020210}212$, $\underline{1202010}212$, $\underline{2012020}212$, $\underline{1201}202012$, $\underline{20210}21212$
$X_8 = 120120212$	$\underline{2120120}212$, $\underline{0210120}212$, $\underline{2102120}212$, $\underline{0210}210212$, $\underline{20210}21212$, $\underline{2120}210212$

The following theorem results directly from the above lemmas.

Theorem 2. *$d_g(s) = n-2$ if and only if fully ternary string s of length n is bad and $d_g(s) = n-3$ otherwise. Moreover, there exists a polynomial time algorithm for grouping ternary strings with a minimum number of flips.*

Proof. The first statement is direct from Lemmas 3 and 4. In case string s is bad, which by Definition 1 can be decided in polynomial time, the algorithm implicit in the proof of Lemma 2 shows how to group s optimally in polynomial time. Otherwise, we repeatedly find a 1-flip to a good string as guaranteed by Lemma 4. The time complexity is $O(n^3)$, since grouping distance, number of choices for a 1-flip, and time to perform a flip and test whether its result is good are all $O(n)$. □

3.2 Grouping Strings over Larger Alphabets

Lemmas 1 and 2 say that $n - k \leq d_g(s) \leq n - 2$ for any fully k-ary string s. For any k there are fully k-ary strings that have flip grouping distance equal to

$n-2$. For example the length $n = 2(k-1)$ string $1020\ldots(k-1)0$ requires for every 1-flip to bring a 0 to the front first and hence we need as many 0-flips as 1-flips, and $d_g(1020\ldots(k-1)0) \geq 2(k-2) = 2k-4 = n-2$. Computer calculations suggest that for $k = 4$ and $k = 5$, for n large enough, the strings with grouping distance $n-2$ are precisely those having identical symbols at every other position, starting from the last (i.e. type II of Definition 1). Proving (or disproving) this statement remains open, as well as finding a polynomial time algorithm for grouping k-ary strings for any fixed $k > 3$. We do, however, have the following intermediate result:

Theorem 3. *For every fixed k there is a PTAS for grouping k-ary strings.*

Proof. Follows from the algorithm in the proof of Lemma 2. We defer the details to a full version of the paper [15]. □

Clearly, there is a strong relationship between grouping and sorting. Understanding grouping may help us to understand sorting, and lead to improved bounds (especially as the length of strings becomes large relative to their arity), because for a k-ary string s, we have $d_g(s) \leq d_s(s) \leq d_g(s) + wc(k)$, with $wc(k)$ the flip diameter on permutations with k elements, as defined before.

Also $d_g(s) = \min\{d_s(t) : t \text{ a relabeling of } s\}$, which gives (for fixed k) a polynomial time reduction from grouping to sorting. Thus every polynomial time algorithm for sorting by prefix reversals directly gives a polynomial time algorithm for the grouping problem (for fixed k).

4 Sorting

In this section we present results on sorting similar to those on grouping in the previous section. Also flip sorting distance remains open for strings over alphabets of size larger than 3. As an intermediate result we thus provide at the end of this section a PTAS for each such problem.

Again a *1-flip* brings identical symbols together and thus shortens the representative of the equivalence class under symbol duplication. But since symbol order matters for sorting, relabelled strings are no longer equivalent. As in grouping, sorting of binary strings is straightforward:

Theorem 4. *$d_s(s) = n-2$ for every fully binary string s of length n with $s_n = 1$, and $d_s(s) = n-1$ otherwise.*

Proof. Exactly $n-2$ 1-flips suffice and are necessary to arrive at length 2 string 01 or 10. If the last symbol is 0 an additional 0-flip is necessary putting a 1 at the end. All these flips can be $f^{(2)}$. □

From Lemma 1 we know that $d_g(s) \geq n-3$ and hence $d_s(s) \geq n-3$ for every ternary string s of length n. In the upper bound on $d_s(s)$ we derive below we focus on strings s ending in a 2 ($s_n = 2$), since sorting distance is invariant under appending a 2 to a string. It turns out that, when sorting a ternary string ending in a 2, one needs at most one 0-flip, except for the string 0212.

Lemma 5. $d_s(s) \leq n-2$ *for every fully ternary string* s *of length* n *with* $s_n = 2$, *except* 0212.

Proof. It is easy to check that 0212 requires 3 flips to be sorted. By induction on n we prove the rest of the lemma. The basis case of $n = 3$ is trivial. For a string s of length $n > 3$ we distinguish three cases:

- $s_{n-1} = 0$: If $s = 20102$ it is sorted in 3 flips: $\underline{20102} \to \underline{0102} \to \underline{102} \to 012$. Otherwise, by induction and relabeling $0 \leftrightarrow 2$, the string $s_1 \dots s_{n-1}$ can be reduced to 210 in $n-3$ flips (to 20 or 10 by Theorem 4 if $s_1 \dots s_{n-1}$ has only two symbols), and one more flip sorts s to 012.
- $s_{n-1} = 1$, $s_1 = 0$ and appears only once: Thus $s = 0(12)^{\geq 2}$ or $s = 02(12)^{\geq 2}$. Then s can be sorted with only one 0-flip: $\underline{0(12)^+12} \to \underline{1(21)^+02} \to \dots \to \underline{2102} \to 012$ or, respectively, $\underline{02}(12)^{\geq 2} \to \underline{20(12)^+12} \to \underline{(12)^+102} \to \dots \to \underline{2102} \to 012$.
- $s_{n-1} = 1$, s_1 not unique:
  If $s = 12012$ then 3 flips suffice: $\underline{12}012 \to \underline{21012} \to \underline{102}12 \to 012$.
  Otherwise, since the other 2 parents of 0212 can flip to 1202, there is a 1-flip to a string $\neq 0212$ to which we can apply the induction hypothesis. □

As in Section 3, we characterise the strings ending in a 2 that need $n-2$ rather than $n-3$ flips to sort.

Definition 2. *We define* bad *strings as all fully ternary strings ending in a* 2 *of the types:*

I. $0(12)^{\geq 2}$
II. $(\{0,1\}2)^+$ *and* $2(\{0,1\}2)^+$
III. $(\{1,2\}0)^+2$ *and* $0(\{1,2\}0)^+$
IV. $(\{1,2\}0)^+12$ *and* $(0\{1,2\})^+012$ *with at least two* 2*s.*
V. $(01)^*0212$ *and* $(10)^+212$
VI. $1(20)^+1(20)^*2$ *and* $0(21)^+0(21)^*2$
VII. $1(02)^+1(02)^+$
VIII. $1(02)^+12$
IX. *77 strings of length at most 11, shown in Table 2.*

All other fully ternary strings ending in a 2 *are* good *strings. Strings of type I-VIII (I-strings ... VIII-strings for short) are called generically bad, or* g-bad *for short.*

This definition makes 0212 a bad string as well. From Lemma 5 we know that 0212 is the only ternary string ending in a 2 with sorting distance $n-1$.

Theorem 5. *String* 0212 *has sorting distance* 3. *Any other fully ternary string* s *of length* n *with* $s_n = 2$ *has prefix reversal sorting distance* $n-2$ *if it is bad and* $n-3$ *if it is good. A fully ternary string* s *ending in a* 0 *or* 1 *has the same sorting distance as* $s2$.

Table 2. Type IX strings

$Y_1 = 210212$	$Y_{17} = 10210212$	$Y_{33} = 12120102$	$Y_{48} = 010210212$	$Y_{63} = 1021201012$
$Y_2 = 021012$	$Y_{18} = 21021212$	$Y_{34} = 12010212$	$Y_{49} = 010210202$	$Y_{64} = 1020210212$
$Y_3 = 212012$	$Y_{19} = 02102012$	$Y_{35} = 12010202$	$Y_{50} = 010212012$	$Y_{65} = 1010210202$
$Y_4 = 120102$	$Y_{20} = 02101212$	$Y_{36} = 20120102$	$Y_{51} = 202010212$	$Y_{66} = 0202010212$
$Y_5 = 201202$	$Y_{21} = 10212012$	$Y_{37} = 12012012$	$Y_{52} = 121202012$	$Y_{67} = 2120202012$
$Y_6 = 0210202$	$Y_{22} = 02121012$	$Y_{38} = 021021202$	$Y_{53} = 121201202$	$Y_{68} = 2120102012$
$Y_7 = 1021202$	$Y_{23} = 02120102$	$Y_{39} = 102120102$	$Y_{54} = 201021202$	$Y_{69} = 2021021212$
$Y_8 = 0212012$	$Y_{24} = 10102102$	$Y_{40} = 102010212$	$Y_{55} = 120212012$	$Y_{70} = 2010212012$
$Y_9 = 2120102$	$Y_{25} = 02010212$	$Y_{41} = 021202012$	$Y_{56} = 012021212$	$Y_{71} = 1201021202$
$Y_{10} = 0102102$	$Y_{26} = 21202012$	$Y_{42} = 021201012$	$Y_{57} = 120102012$	$Y_{72} = 1201202012$
$Y_{11} = 1212012$	$Y_{27} = 21201012$	$Y_{43} = 020210212$	$Y_{58} = 201202012$	$Y_{73} = 10202010212$
$Y_{12} = 2010212$	$Y_{28} = 21201202$	$Y_{44} = 101020212$	$Y_{59} = 120120212$	$Y_{74} = 02120102012$
$Y_{13} = 0120212$	$Y_{29} = 20210212$	$Y_{45} = 020212012$	$Y_{60} = 201201012$	$Y_{75} = 02021021212$
$Y_{14} = 1201012$	$Y_{30} = 01021202$	$Y_{46} = 212010202$	$Y_{61} = 0210212012$	$Y_{76} = 21201202012$
$Y_{15} = 1201212$	$Y_{31} = 01020212$	$Y_{47} = 212012012$	$Y_{62} = 1021202012$	$Y_{77} = 12120202012$
$Y_{16} = 2012012$	$Y_{32} = 20212012$			

Proof. Directly from Lemmas 6 and 7 below. Note that every sorting sequence for s sorts $s2$ as well while every sorting sequence for $s2$ can be modified to avoid flipping the whole string and thus works for s as well. □

Lemma 6. $d_s(s) = n - 2$ *for every bad ternary string* $s \neq 0212$ *of length* n.

Proof. Since $d_s(s) \geq n - 3$ and any 1-flip decreases the length of the string by 1, Lemma 5 says it suffices to show that for each type in Definition 2 a 0-flip is necessary.

- For I-strings only 0-flips are possible.
- A 1-flip on a II- or III-string leads to a string of the same type, so that eventually no 1-flip is possible.
- A 1-flip on a IV-string leads either again to a IV-string or (when destroying the 12 suffix) to a III-string.
- A 1-flip on a V-string leads either again to a V-string or (when destroying the suffix with a $\underline{\ldots 02}12$ flip) to a IV-string. Flips $\underline{\ldots 0}212$ and $\underline{\ldots 0212}$ are not possible for lack of more 2's.
- For strings of VI-, VII- and VIII-strings only one 1-flip is possible, leading to II-, III- and IV-strings respectively.
- For IX-strings it is easy (although time consuming) to check that every 1-flip either leads to a shorter IX-string or to a string of type I-VIII [15]. □

Lemma 7. $d_s(s) = n - 3$ *for every good ternary string* s *of length* n.

Proof. The proof is by induction on n and is similar to the proof of Lemma 4. We defer the details to a full version of the paper [15]. □

Theorem 6. *There exists a polynomial time algorithm for optimally sorting ternary strings.*

Proof. Follows rather easily from Theorem 5. □

Finally, in light of the fact that the complexity of the sorting problem on quaternary (and higher) strings remains open, the following serves as an intermediate result:

Theorem 7. *For every fixed k there is a PTAS for sorting k-ary strings.*

Proof. The proof is very similar to that used in Theorem 3. The details are again deferred to a full version of the paper [15]. □

5 Prefix Reversal Diameter

Let $S(n,k)$ be the set of fully k-ary strings of length n. We define $\delta(n,k)$ as the largest value of $d(s,t)$ ranging over all compatible $s,t \in S(n,k)$. The value $\delta(n,k)$ is called the *prefix reversal diameter* of fully k-ary, length-n strings.

Theorem 8. *For all $n \geq 2$, $\delta(n,2) = n-1$.*

Proof. To prove $\delta(n,2) \geq n-1$, consider compatible $s,t \in S(n,2)$ with $s = (10)^{n/2}$ in case n even and $s = 0(10)^{(n-1)/2}$ in case n odd and in both cases $t = I(s)$ i.e. t is the sorted version of s. By Theorem 4, $d(s,t) \geq n-1$.

The proof that $\delta(n,2) \leq n-1$, for all $n \geq 2$ is by induction on n. The lemma is trivially true for $n = 2$. Consider two compatible binary strings of length n: $s = s_1 s_2 \ldots s_n$ and $t = t_1 t_2 \ldots t_n$. If $s_n = t_n$ then by induction $d(s,t) \leq n-2$. Thus, suppose (wlog) $s_n = 0$ and $t_n = 1$. If $t_1 = 0$ then $f^{(n)}t$ and s both end with a 0, and using induction and symmetry $d(s,t) \leq 1 + d(f^{(n)}t, s) \leq n-2+1 = n-1$. An analogous argument holds if $s_1 = 1$.

Remains the case $s_1 = s_n = 0$ and $t_1 = t_n = 1$. First, suppose $t_{n-1} = 0$. Since s and t are compatible, there must exist index i such that $s_i = 0$ and $s_{i+1} = 1$. Hence, $f^{(n)}(f^{(i+1)}(s))$ ends with 01 like t and by induction $d(s,t) \leq 2 + d(f^{(n)}(f^{(i+1)}(s)), t) = 2 + n - 3$. Analogously, we resolve the case $s_{n-1} = 1$.

Finally, suppose $s = 0...00$ and $t = 1...11$. If s contains 11 as a substring, then flipping that 11 (in the same manner as above) to the back of s using 2 flips, gives two strings that both end in 11. Alternatively, if s does not contain 11 as a substring then s has at least two more 0's than 1's, which implies that t must contain 00 as a substring. In that case two prefix reversals on t suffice to create two strings that both end with 00. In both cases, the induction hypothesis gives the required bound. □

Note that, trivially, $d(s,t) \leq 2n$ for all compatible $s,t \in S(n,k)$, for all k, because two prefix reversals always suffice to increase the maximal common suffix between s and t by at least 1. The following tighter bound gives the best bound known on the diameter of ternary strings.

Lemma 8. *For any two compatible $s,t \in S(n,k)$, for any k, let a be the most frequent symbol in s and α its multiplicity. Then $d(s,t) \leq 2(n-\alpha)$.*

Proof. We prove the lemma, by induction on n. The lemma is trivially true for $n = 2$. Consider $s,t \in S(n,k)$. If $s_n = t_n = a$ then $s_1 s_2 \ldots s_{n-1}$ and $t_1 t_2 \ldots t_{n-1}$

are compatible length-$(n-1)$ strings where the most frequent symbol occurs at least $\alpha-1$ times. Thus, by induction $d(s,t) \leq 2((n-1)-(\alpha-1)) = 2(n-\alpha)$. In case $s_n = t_n \neq a$ induction even gives $d(s,t) \leq 2((n-1)-(\alpha)) = 2(n-\alpha)-2$. Thus, suppose $s_n \neq t_n$ implying (wlog) that $t_n = b \neq a$. Suppose $s_i = b$; after two flips $s' = f^{(n)}(f^{(i)}(s))$ has b at the end; $s'_n = t_n$. Moreover the length $n-1$ suffixes of s' and t still contain α a's. Hence by induction $d(s,t) \leq 2 + d(s',t) \leq 2 + 2((n-1)-\alpha) = 2(n-\alpha)$. □

Lemma 9. *For all $n > 3$, $n-1 \leq \delta(n,3) \leq (4/3)n$.*

Proof. Since in any ternary case $\alpha \geq \lceil n/3 \rceil$, Lemma 8 implies $\delta(n,3) \leq (4/3)n$. To prove $\delta(n,3) \geq n-1$ we distinguish between n is odd and n is even. For odd $n = 2h+1$, let s be $2(01)^h$, and for even $n = 2h$ let $s = 01(21)^{h-1}$. In both cases we let $t = I(s)$. We observe that, in the even and in the odd case, $s2$ is a bad I-string and a bad IV-string, respectively, in the sense of Definition 2. Thus, by Theorem 5 we have that $d(s,t) = d(s2,t2) = (n+1) - 2 = n-1$. (Here $s2$, respectively $t2$, refers to the concatenation of s, respectively t, with an extra 2 symbol.) □

Brute force enumeration has shown that, for $4 \leq n \leq 13$, $\delta(n,3) = n-1$. (Note that $\delta(3,3) = 3$ because $d(021,012) = 3$.) Proving or disproving the conjecture that $\delta(n,3) = n-1$ for $n > 3$ remains an intriguing open problem[3].

6 Prefix Reversal Distance

We show that computing flip distance is NP-hard on binary strings. We also point out, using a result from [17], that computing flip distance on *arbitrary* strings is polynomial-time reducible (in an approximation-preserving sense) to computing it on binary strings.

Theorem 9. *The problem of computing the prefix reversal distance of binary strings is NP-hard.*

Proof. We prove NP-completeness of the corresponding decision problem:

Name: BINARY-PD (2PD shortly)
Input: Two compatible strings $s,t \in S(n,2)$, and a bound $B \in \mathbb{Z}^+$.
Question: Is $d(s,t) \leq B$?

2PD∈NP, since a certificate for a positive answer consists of at most B flips[4]. To show completeness we use a reduction from 3-PARTITION [10] (cf. [4] and [17]).

Name: 3-PARTITION (3P shortly)
Input: A set $A = \{a_1, a_2, ..., a_{3k}\}$ and a number $N \in \mathbb{Z}^+$. Element a_i has size $r(a_i) \in \mathbb{Z}^+$ satisfying $N/4 < r(a_i) < N/2$, $i = 1, \ldots, 3k$, and $\sum_{i=1}^{3k} r(a_i) = kN$.

[3] Interestingly, initial experiments with brute force enumeration have also shown that, for $4 \leq n \leq 10$, $\delta(n,4) = n$, and for $5 \leq n \leq 9$, $\delta(n,5) = n$.
[4] Recall that for all compatible strings $s,t \in S(n,2)$, trivially $d(s,t) \leq 2n$.

Question: Can A be partitioned into k disjoint triplet sets A_1, A_2, ..., A_k such that $\sum_{a \in A_j} r(a) = N$, $j = 1, \ldots, k$?

Given instance $I = (A, N, r)$ of 3P, we create an instance of 2PD by setting $B = 6k$ and building two compatible binary strings s and t:

$$s = \left(\prod_{1 \leq i \leq 3k} 0001^{r(a_i)} \right) 000 \qquad t = 0^{3(3k+1)-k} (01^N)^k$$

This construction is clearly polynomial in a unary encoding of the 3P instance; we use the *strong* NP-hardness of 3P [10]. We claim that $I = (A, N, r)$ is a positive instance of 3P if and only if $d(s, t) \leq 6k$. We defer the proof to a full version of the paper [15]. □

For studying problems on arbitrary strings, let X and Y be two compatible, length-n strings, where we assume (wlog) that each of the symbols from X and Y are drawn from the set $\{0, 1, ..., n-1\}$. We define $D(X, Y)$ as the smallest number of flips required to transform X to Y. The arity of the strings X and Y does not need to be fixed, and symbols may be repeated. Hence, sorting of a permutation by flips (MIN-SBPR), and the flip distance problem over fixed arity strings, are both special cases of computing D. Given that computing D is a generalisation of computing distance d of binary strings, immediately implies that it is NP-hard. However, an approximation-preserving reduction in the *other* direction is possible, meaning that inapproximability results for one of the problems will be automatically inherited by the other.

Theorem 10. *Given two compatible strings X and Y of length n with each symbol from X and Y drawn from $\{0, 1, ..., n-1\}$, it is possible to compute in time polynomial in n two binary strings x and y of length polynomial in n such that $D(X, Y) = d(x, y)$.*

Proof. This result follows directly from work by Radcliffe, Scott and Wilmer [17]. The proof is deferred to a full version of the paper [15]. □

7 Open Problems

In this study we have unearthed many rich (and surprisingly difficult) combinatorial questions which deserve further analysis. We discuss some of them here. The main unifying, "umbrella" suggestion is that, to go beyond ad-hoc (and case-based) proof techniques, it will be necessary to develop deeper, more structural insights into the action of flips on strings over fixed size alphabets.

Grouping and sorting on higher arity alphabets. We have shown how to group and sort optimally binary and ternary strings, but characterisations and algorithms for quaternary (and higher) alphabets have so far evaded us. As observed in Section 3.2, it *seems* that for $k = 4, 5$ and for sufficiently long strings, the strings with grouping distance $n-2$ settle into some kind of pattern,

but this has not yet offered enough insights to allow either the development of a characterisation or of an algorithm. Related problems include: for all fixed k, are there polynomial algorithms to optimally sort (optimally group) k-ary strings? Is grouping strictly easier than sorting, in a complexity sense? How does grouping function under other operators e.g. reversals, transpositions? An upper bound on the grouping transposition distance has been presented in [7].

Diameter questions. Proving or disproving that $\delta(n, 3) = n - 1$ for $n > 3$ remains the obvious open diameter question. Beyond that, diameter results for quaternary and higher arity alphabets are needed. How does the diameter $\delta(n, k)$ grow for increasing k? (At this point we conjecture that, for sufficiently long strings, the diameter of 3-ary, 4-ary and 5-ary strings is $n - 1$, n, and n respectively.)

The suspicion also exists that, for all k and for all sufficiently long n, there exists a length-n fully k-ary string s such that $d(s, I(s)) = \delta(n, k)$. In other words, the set of all pairs of strings that are $\delta(n, k)$ flips apart includes some instances of the sorting problem. It should be noted however that, following empirical testing, it is apparent that there are also very many pairs of strings s, t with $s \neq I(t)$ and $t \neq I(s)$ that are $\delta(n, k)$ flips apart.

It also seems important to develop diameter results for subclasses of strings, perhaps (as in [17]) characterised by the frequency of their most frequent symbol. It may be that such refined diameter results for k-ary alphabets provide information that is important in determining $\delta(n, k + 1)$.

Note finally that the diameter of strings over fixed size alphabets, i.e. $\delta(n, k)$, is always bounded from above by the diameter of permutations. This is because the distance problem on two length-n, fixed size alphabet strings s, t can easily be re-written as a sorting problem on a length-n permutation π, such that a sequence of prefix reversals sorting the permutation also suffices to transform s into t. Indeed, because of this relabelling property, the flip distance between two fixed size alphabet strings can be viewed as being equal to the minimum permutation sorting distance, ranging over all such relabellings into a permutation. Can this relationship between the fixed size alphabet and permutation world be further specified and exploited?

Signed strings. The problem of sorting signed permutations by flips (the *burnt* pancake flipping problem) is well known [5] [11] [14], but in this paper we have not yet attempted to analyse the action of flips on signed, fixed size alphabet strings. Obviously, analogues of all the problems described in this paper exist for signed strings.

Complexity/approximation. In the presence of hardness results (e.g. Theorem 9) it is interesting to explore the complexity of restricted instances, and to develop algorithms with guaranteed approximation bounds. For example, [17] gives a PTAS for dense instances. The development of approximation algorithms is also a useful intermediate strategy where the complexity of a problem remains elusive. In particular, this requires the development of improved lower bounds.

References

1. Bader, M., Ohlebusch, E.: Sorting by weighted reversals, transpositions, and inverted transpositions. In: Apostolico, A., Guerra, C., Istrail, S., Pevzner, P., Waterman, M. (eds.) RECOMB 2006. LNCS (LNBI), vol. 3909, pp. 563–577. Springer, Heidelberg (2006)
2. Bergeron, A., Mixtacki, J., Stoye, J.: On sorting by translocations. J. Comp. Biol. 13(2), 567–578 (2006)
3. Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., Jiang, T.: Assignment of orthologous genes via genome rearrangement, IEEE/ACM Trans. Comput. Biol. Bioinform. 2(4) (October- December 2005)
4. Christie, D.A., Irving, R.W.: Sorting strings by reversals and transpositions. SIAM J. on Discrete Math. 14(2), 193–206 (2001)
5. Cohen, D.S., Blum, M.: On the problem of sorting burnt pancakes. Discrete Appl. Math. 61(2), 105–120 (1995)
6. Elias, I., Hartman, T.: A 1.375-approximation algorithm for sorting by transpositions. IEEE/ACM Trans. Comput. Biology Bioinform. 3(4), 369–379 (2006)
7. Eriksson, H., Eriksson, K., Karlander, J., Svensson, L., Wastlund, J.: Sorting a bridge hand. Discrete Math. 241, 289–300 (2001)
8. Fischer, J., Ginzinger, S.W.: A 2-approximation algorithm for sorting by prefix reversals. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 415–425. Springer, Heidelberg (2005)
9. Garey, M.R., Johnson, D.S.: Complexity results for multiprocessor scheduling under resource constraints. SIAM J. Comput. 4(4), 397–411 (1975)
10. Garey, M.R., Johnson, D.S.: Computers And Intractability: A Guide To The Theory Of NP-completeness. W.H. Freeman, San Francisco, CA (1979)
11. Gates, W.H., Papadimitriou, C.H.: Bounds for sorting by prefix reversal. Discrete Math. 27, 47–57 (1979)
12. Goldstein, A., Kolman, P., Zheng, J.: Minimum Common String Partition problem: hardness and approximations, Electron. J. Combin. 12, #R50 (2005)
13. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip: a polynomial algorithm for sorting permutations by reversals. Jour. ACM 46(1), 1–27 (1999)
14. Heydari, M.H., Sudborough, I.H.: On the diameter of the pancake network. J. Algorithms 25, 67–94 (1997)
15. Hurkens, C.A.J., van Iersel, L.J.J., Keijsper, J.C.M., Kelk, S.M., Stougie, L., Tromp, J.T.: Prefix reversals on binary and ternary strings, technical report (2006), `http://www.win.tue.nl/bs/spor/2006-10.pdf`
16. Morales, L., Sudborough, I.H.: Comparing Star and Pancake Networks. In: The Essence Of Computation: Complexity, Analysis, Transformation, LNCS (2002), Springer, Heidelberg, pp. 18–36 (2566)
17. Radcliffe, A.J., Scott, A.D., Wilmer, E.L.: Reversals and transpositions over finite alphabets. SIAM J. on Discrete Math. 19(1), 224–244 (2005)
18. Scott, A.D.: Personal communication