Information Processing Letters 132 (2018) 44–48

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Sorting signed permutations by reversals using link-cut trees

Irena Rusu

LS2N, UMR 6004, Université de Nantes, 2 rue de la Houssinière, BP 92208, 44322 Nantes, France

A R T I C L E I N F O

Article history:
Received 6 March 2017
Received in revised form 18 October 2017
Accepted 15 December 2017
Available online 20 December 2017
Communicated by P. Wong

Keywords:
Algorithms
Sorting by reversals
Link-cut tree

A B S T R A C T

E. Tannier, A. Bergeron and M.-F. Sagot proposed an algorithm to sort a signed permutation P by performing a minimum number of reversals, for which two implementations exist. With a partition of P in blocks of size b, these implementations take $O(n(\frac{n}{b}\log b + b))$ and respectively $O((\frac{n}{b})^2 + n(\log(\frac{n}{b}) + \frac{n}{b} + b))$ time, where n is the size of the permutation. The best running times of $O(n\sqrt{n\log n})$ and respectively $O(n\sqrt{n})$ are obtained with $b = \sqrt{n\log n}$ and $b = \sqrt{n}$ respectively.
Seeking an $O(n\log n)$ algorithm requires to drop the b addend in the running time formulas, which prevents the choice of a large b. To this end, we propose an implementation of the algorithm whose originality lies in the use of $O(\log n)$ aggregate operations allowed by link-cut trees, and by their filiform variant called log-lists. The resulting algorithm has the advantage of reaching a running time of $O(n(\frac{n}{b}\log b))$, but also has the drawback of potentially making use of very large numbers, reducing in practice the choices of b.

1. Introduction

In bioinformatics, a genome with no duplicate gene is represented as a permutation P on a set of elements $[n] := \{1, 2, \dots, n\}$. The permutation is *signed* when the orientation of each gene, *i.e.* of each element of the permutation, is indicated by a sign "+" (usually dropped) or "−". A *reversal* is one of the operations allowing to rearrange genomes, defined as the left–right inversion of the elements in an interval of the permutation, together with the flip of all signs in the interval. Sorting a signed permutation P by reversals seeks at estimating the evolutionary distance between the genome it represents and another genome, represented by the identity permutation *Id*, by finding the minimum number of reversals allowing to transform P into *Id*.

In this paper we consider signed permutations denoted $P = (0\,P_1 \dots P_n\,n+1)$ over $[n] \cup \{0, n+1\}$, meaning that elements $P_0 = 0$ and $P_{n+1} = n+1$ are respectively added at the beginning and end of every permutation on $[n]$. We formally define the *reversal* $\rho(i, j)$ *of the interval* $[i, j]$ *of* P with $1 \le i \le j \le n$ as the operation transforming P into $(0\,P_1 \dots P_{i-1} {-P_j} {-P_{j-1}} \dots {-P_i}\ P_{j+1} \dots P_n n+1)$. The problem of sorting a signed permutation by reversals has been widely studied (see [3]). The first algorithm with a running time under $O(n^2)$ was proposed in [10] by Tannier, Bergeron and Sagot (therefore we will call it the TBS algorithm). Its first implementation, with running time of $O(n\sqrt{n\log n})$, was reached using a data structure proposed in [6]. Another implementation proposed in [4] yielded a $O(n\sqrt{n})$ running time. The next challenge (which seems reachable, see [9]) is to know whether an $O(n\log n)$-time algorithm exists for the problem, as is the case for other similar problems (see [7]).

A better running-time for the TBS algorithm should reach a (challenging) $O(\log n)$ time for performing each reversal. With this aim, we propose to use a data structure called a log-list, that we introduced in [7] and which is a filiform variant of the very efficient link-cut trees intro-

E-mail address: Irena.Rusu@univ-nantes.fr.

Step	Reversal	Resulting P	π	S_1	S_2	V
(0)		$0\ \ 3\ \ 1\ \ -2\ \ 4$	$0^+3^-3^+1^-1^+2^+2^-4^-$	empty	empty	$\{(0^+,1^-),(\mathbf{1^+},\mathbf{2^-}),(\mathbf{2^+},\mathbf{3^-}),(3^+,4^-)\}$
(1)	$\rho_\pi(1)$	$0\ \ 3\ \ 1\ \ \mathbf{2}\ \ 4$	$0^+3^-3^+1^-1^+\mathbf{2^-}\mathbf{2^+}4^-$	$\rho_\pi(1)$	empty	$\{(0^+,1^-),(2^+,3^-),(3^+,4^-)\}$
(2)	$\rho_\pi(1)$	$0\ \ 3\ \ 1\ \ \mathbf{-2}\ \ 4$	$0^+3^-3^+1^-1^+\mathbf{2^+}\mathbf{2^-}4^-$	empty	$\rho_\pi(1)$	$\{(0^+,1^-),(\mathbf{2^+},\mathbf{3^-}),(3^+,4^-)\}$
(1)	$\rho_\pi(2)$	$0\ \ \mathbf{-1}\ \ \mathbf{-3}\ \ -2\ \ 4$	$0^+\mathbf{1^+}\mathbf{1^-}\mathbf{3^+}\mathbf{3^-}2^+2^-4^-$	$\rho_\pi(2)$	$\rho_\pi(1)$	$\{(\mathbf{0^+},\mathbf{1^-}),(\mathbf{3^+},\mathbf{4^-})\}$
(1)	$\rho_\pi(0)$	$0\ \ \mathbf{1}\ \ -3\ \ -2\ \ 4$	$0^+\mathbf{1^-}\mathbf{1^+}3^+3^-2^+2^-4^-$	$\rho_\pi(2), \rho_\pi(0)$	$\rho_\pi(1)$	$\{(\mathbf{3^+},\mathbf{4^-})\}$
(1)	$\rho_\pi(3)$	$0\ \ 1\ \ \mathbf{2}\ \ \mathbf{3}\ \ 4$	$0^+1^-1^+\mathbf{2^-}\mathbf{2^+}\mathbf{3^-}\mathbf{3^+}4^-$	$\rho_\pi(2), \rho_\pi(0), \rho_\pi(3)$	$\rho_\pi(1)$	$\emptyset$
(1)	$\rho_\pi(3)$	$0\ \ 1\ \ \mathbf{-3}\ \ \mathbf{-2}\ \ 4$	$0^+1^-1^+\mathbf{3^+}\mathbf{3^-}\mathbf{2^+}\mathbf{2^-}4^-$	$\rho_\pi(2), \rho_\pi(0)$	$\rho_\pi(1)$	$\emptyset$
(3)						Return S_1, S_2, that is $\rho_\pi(2), \rho_\pi(0), \rho_\pi(1)$

Fig. 1. The TBS algorithm for $P = (0\,3\,1\,{-2}\,4)$. Oriented pairs, as well as the reversals performed, are in bold.

duced by Sleator and Tarjan [8]. Whereas the bottleneck of our implementation of the TBS algorithm is the potential use of large numbers, we believe that our approach has the merit of showing an original (and probably improvable) way of dealing with signed reversals.

2. Main definitions and the TBS algorithm

A pair (P_i, P_{i+1}), $0 \le i \le n$, is an *adjacency* if $P_i + 1 = P_{i+1}$, otherwise it is a *breakpoint*. The *extended permutation* of P is the signed permutation $\pi(P)$ (or π for short) with $2n+2$ elements, obtained from P by:

- replacing element 0 by element 0^+ and element $n+1$ by element $(n+1)^-$,
- for each $p \ne 0, n+1$, replacing each element $+p$ by the pair of elements $p^-\,p^+$ and each element $-p$ by the pair of elements $p^+\,p^-$.

Adjacencies of P are thus represented in π by pairs of elements $(p^+, (p+1)^-)$ that are next to each other (whatever the order). Within π we identify *oriented pairs* (respectively *unoriented pairs*), which are pairs $(p^+, (p+1)^-)$ such that p and $p+1$ have different (respectively equal) signs in P. Any reversal $\rho(i, j)$ on P is simulated by the reversal $\rho(2i-1, 2j)$ on π, and vice-versa.

As the aim of sorting by reversals is to obtain $\pi(Id)$ from π by performing a minimum number of reversals, each reversal seeks to bring p^+ (for some p) next to $(p+1)^-$ by a reversal on π which may be seen as corresponding to the oriented pair $(p^+, (p+1)^-)$. This reversal, denoted $\rho_\pi(p)$, is defined as the reversal $\rho(2i-1, 2j)$ on π spanning all the pairs k^+k^- or k^-k^+ contained in π between p^+ (included) and $(p+1)^-$ (included).

The TBS algorithm [10] (except its part clearing the connected components, solved in [5] in almost linear time) is below. Fig. 1 shows an example.

(0) Let $V = \{(p^+, (p+1)^-) \mid \text{non-adjacent pair}\}$. Let S_1, S_2 be empty sequences of reversals.
(1) While there is an oriented pair $(p^+, (p+1)^-)$ in V, add $\rho_\pi(p)$ at the end of S_1, perform the reversal and remove the resulting adjacencies (one or two) from V. If the pair corresponding to the first reversal of S_2 is not oriented, then perform the latter reversal back and remove it from S_1 (but do not update V).
(2) If V is not empty, then re-apply the reversals from S_1 in reverse order (but do not update V) until one of the pairs in V becomes oriented. While doing this, move each performed reversal from the end of S_1 to the beginning of S_2. Goto step (1).
(3) Return the sequence S_1, S_2.

The implementations of the TBS algorithm proposed in [10] (using the data structure proposed by Kaplan and Verbin in [6]) and in [4] both cut the permutation into contiguous blocks of size b, use an underlying data structure allowing to reach a given block in logarithmic time of the number of blocks, and an inner data structure to store, split, reverse and concatenate blocks. The differences come from the management of oriented/unoriented pairs, which is done within each block for Kaplan and Verbin's data structure, and is distributed inside and outside the blocks, at each level of the underlying structure, for Han's approach. As a result, the running times of the algorithms are $O(n(\frac{n}{b}\log b + b))$ (Kaplan and Verbin's approach) and respectively $O((\frac{n}{b})^2 + n(\log(\frac{n}{b}) + \frac{n}{b} + b))$ (Han's approach). The choices of $b = \sqrt{n\log n}$ and respectively of $b = \sqrt{n}$ allow to equilibrate all the terms in each sum, yielding $O(n\sqrt{n\log n})$ and respectively $O(n\sqrt{n})$ best running times.

In both these approaches, the b addend in the complexity formula looks like a failure. The inner data structure in each block is made so as to allow a quick update of the oriented/unoriented pairs once a reversal is performed, but not to cut a block by separating its elements smaller or larger than a given value. Therefore, when a reversal has to be performed, the two blocks containing the endpoints of the reversal are cut by successively considering each element of each block, thus in $O(b)$ time.

We propose here a different approach, aiming at dropping the b addend in the complexity formula by: proposing a partition of π in non-contiguous blocks, containing both endpoints of each pair affected to the block; affecting weights to the endpoints of each pair $(p^+, (p+1)^-)$ such that a pair has non-zero sum of its two weights iff the pair is oriented; choosing the weights so that the total sum of the elements in each block indicates the existence and, if so, the localization of an oriented pair.

3. Outline of our implementation

Link-cut trees [8] aim at efficiently maintaining a forest of vertex-disjoint dynamic rooted trees with weighted arcs under cut and link operations. They allow to perform in logarithmic time a long list of operations, among which aggregate operations on paths. As a compensation, usual operations like getting the weight of an arc or modifying it also need logarithmic time. Note that, depending