

Easy ways to clear hurdles

Anne Bergeron

LaCIM, Université du Québec à Montréal, Canada.

bergeron.anne@uqam.ca

Abstract

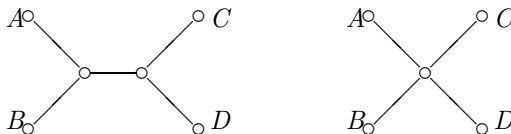
One of the puzzling aspect of the Hannenhalli-Pevzner theory is the complex, but always colorful, classification of *hurdles*. In this paper, we show that simple results on trees are at the root of all results on hurdles, either maximal or simple, super-hurdles, and fortresses.

1 Pruning trees

Let T be an unrooted tree with more than two leaves. Define a *branch* in T to be a segment from a leaf, up to the next fork, but excluding it. An *even pruning* of a tree T is the simultaneous cutting of two branches that **does not** create new leaves.

Proposition 1 *Any tree with at least four leaves can be evenly pruned.*

Proof: Any four leaves are related as in one of the following diagrams. In each case, pruning the set of branches $\{A, D\}$, or the set $\{B, C\}$, never creates new leaves.



Proposition 2 *A tree with three leaves cannot be evenly pruned.*

A *long branch* is a branch that contains more than one node. We are interested in destroying trees using even pruning and branch cutting. The cost of even pruning and cutting long branches is set to 2, and the cost of cutting a leaf is set to 1. If a tree contains only 1 or 2 leaves, then the cost of destroying it is, respectively, 1 and 2. Define the *cost* of a tree to be the sum of costs of operations necessary to destroy it. We have the following:

Corollary 1 *The minimal cost of a tree with $2k$ leaves is $2k$. The minimal cost of a tree with $2k + 1$ leaves, all of them in long branches, is $2k + 2$, otherwise it is $2k + 1$.*

Note: For those familiar with the Hannenhalli-Pevzner terminology [2], the number of leaves of T corresponds to the number of *hurdles*, and long branches to *super-hurdles*. Even pruning, and cutting long branches, correspond to *hurdle merging*, and cutting a leaf corresponds to *hurdle cutting*. A tree with an odd number of long branches is a *fortress*. Corollary 1 thus translates as: “If there are $2k$ hurdles, the reversal distance is $d = n - c + 2k$. If there are $2k + 1$ hurdles, all of which are super-hurdles, then $d = n - c + 2k + 2$, otherwise $d = n - c + 2k + 1$ ”.

2 Clearing hurdles

Let π be a permutation on n elements whose first element is 1, and last element is n . A *component* of π is an interval $[a, a + i], i \geq 2$ whose set of elements is $\{a, \dots, a + i\}$, and is not the union of two such intervals. For example, the permutation in Fig. 1 has 5 components: $[1, 13], [13, 16], [3, 9], [9, 12]$ and $[5, 8]$.

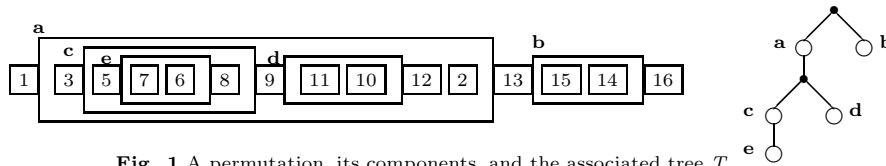


Fig. 1 A permutation, its components, and the associated tree T

It is shown in [1] that components are either nested with different endpoints, linked with one endpoint, or disjoint. We associate a tree T to the components of a permutation in the following way: 1) Create a node for each component; 2) Join all components that are linked together to a single new node; 3) Join a component, or a group of linked components, to the component in which it is immediately nested.

In the Hannenhalli-Pevzner theory, a crucial process in computing the reversal distance to *erase* all components of a positive permutation. Erasing a single component can be done a cost 1, or a group of components can be erased by merging two components, at cost 2. The components that are erased by merging two components c_1 and c_2 include, among others, any component that contains either c_1 and c_2 , but not both. We have:

Proposition 3 *Merging two leaves c_1 and c_2 in T can be simulated by cutting the two branches associated with c_1 and c_2 .*

Corollary 2 *The minimal cost of erasing all components of a permutation is given by Corollary 1.*

This result depends only on the fact that merging two leaves erases any component that contains either one but not both, which is easy to prove. However, if one wants to characterize all the components that are erased when merging two components, the tree T is useful, since one can prove that:

Proposition 4 *Components erased by merging two components c_1 and c_2 are the components along the unique path that joins c_1 and c_2 in T .*

In the above example, merging components **e** and **d** erases **c**, but not **a**. If **c** and **d** were not linked, they would both be direct descendant of node **a**, and component **a** would be erased by the merging of **e** and **d**.

References

- [1] Anne Bergeron and Jens Stoye, *On the Similarity of Sets of Permutations and its Applications to Genome Comparison*, LNCS 2697, Springer-Verlag, Berlin: 68-79 (2003).
- [2] Sridhar Hannenhalli and Pavel Pevzner, *Transforming Cabbage into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals*. JACM 46(1): 1-27 (1999).