Overview

- The problem - comparing shapes.
- Skeletons or “Shock Graphs”.
- Application of Edit distance here.
- Edit distance algorithm.

Problem : Comparing Shapes

Simple closed curves in a plane, i.e *no holes*.

General Approach:

1. Represent a shape by its "skeleton", a graph.
   (Vision community has techniques for doing so.)

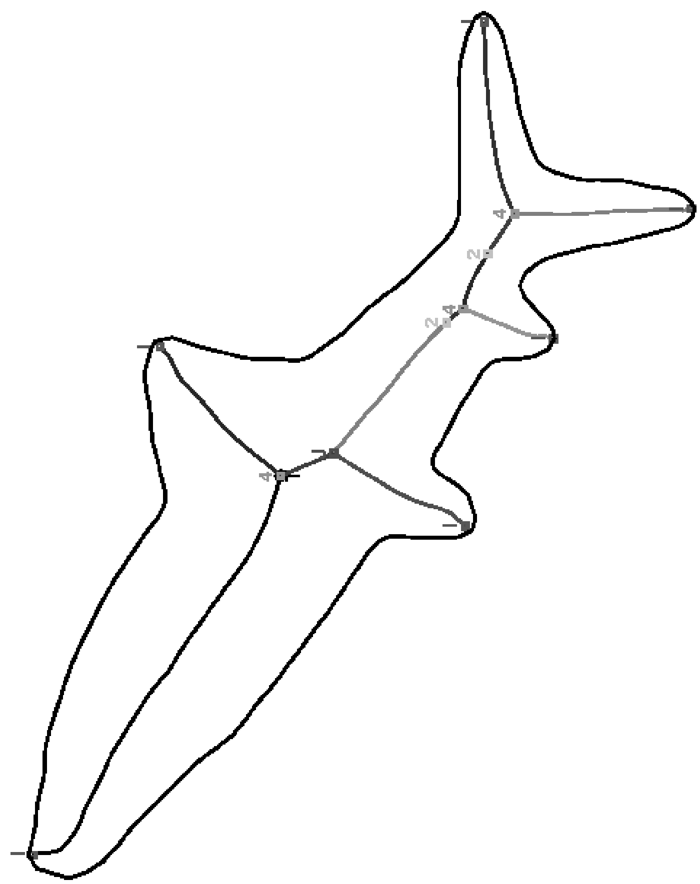
2. Compare the skeletons using edit distance.

Shock Graphs

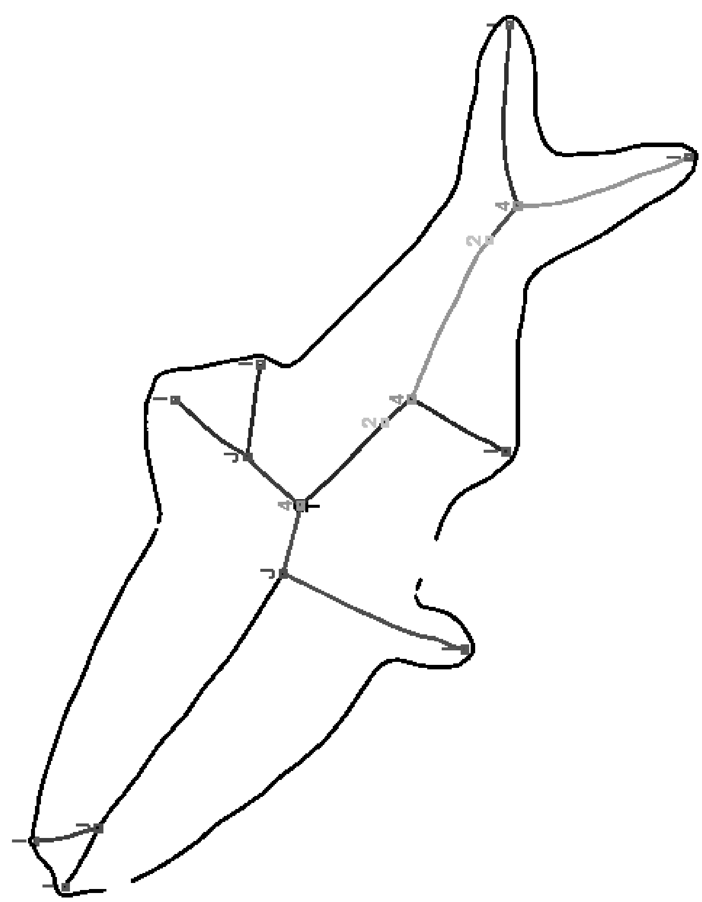
- Shock Graphs are constructed from the *locus of centers of maximal circles at least bitangent to the boundary*.

- Convert into a combinatorial object.
  Now the arcs of this graph have attributes like:
  - *Radius* - the distance from the boundary.
  - *Velocity* - rate of change of radius.

Shock Graph example



Shock Graph example (2)



Edit Distance

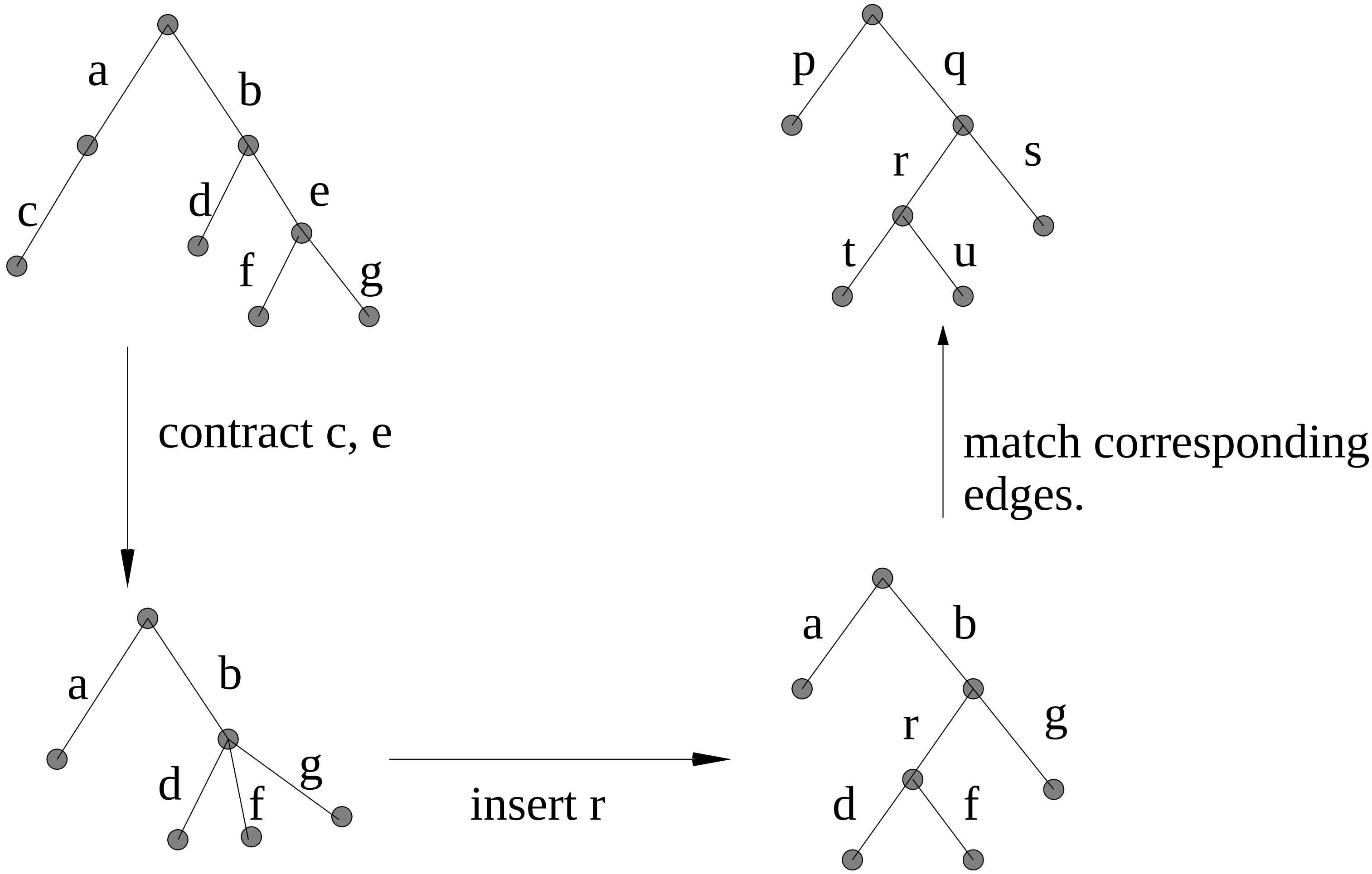
- Observation : These graphs are trees if the shapes are *simple* and *closed*.

- Idea : Use tree edit distance to compare shapes.

Edit distance between two trees T_1 and T_2 is *the minimum cost of a sequence of "edit operations" that takes tree* T_1 *to tree* T_2.

Traditionally, edit operations are *Edge insertion, Edge deletion, Matching an edge in* T_1 *to one in* T_2.

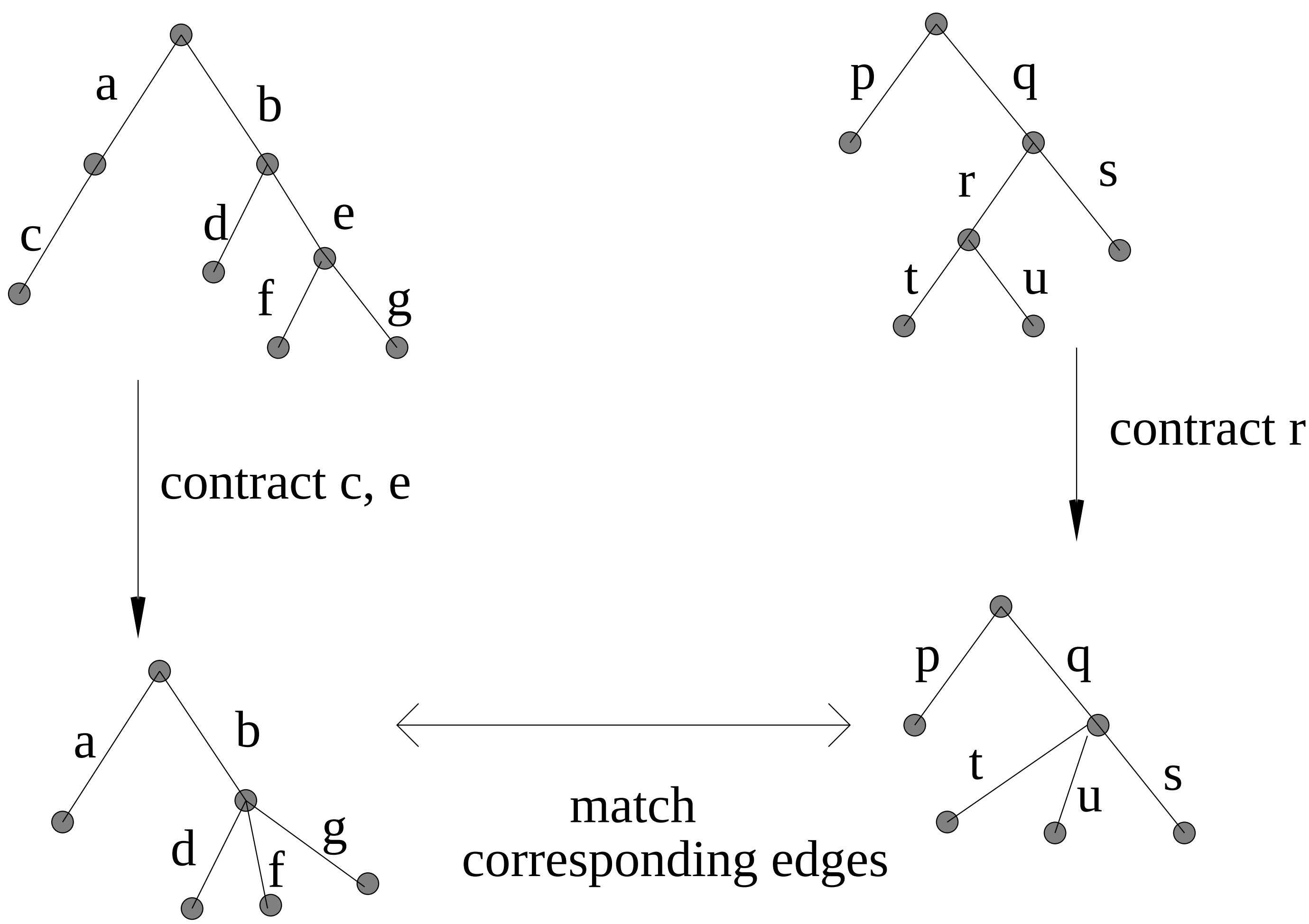
Example of an edit sequence



Tree Edit Distance - contd

An alternative (equivalent) definition which we will use:
Edit distance is the minimum cost of separately transforming the two input trees into a common tree.

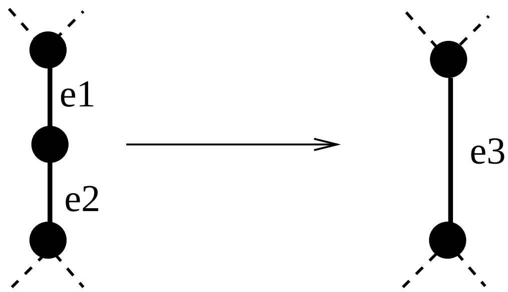
We now don't need the insert (or uncontract) operation.



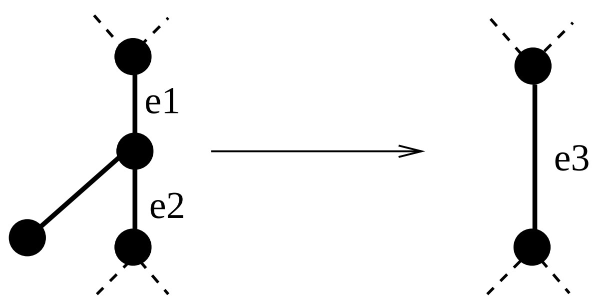
The twist in our problem

We require a more general set of edit operations.

Merge : Combine two edges with a common endpoint (of degree two) into a single edge.



Prune : If common endpoint has degree three, merge is still allowed. The subtree rooted at the other incident edge is *pruned* off.



Considered natural and essential edit operations by the vision researchers (Kimia and Sharvit).

Problem Definition

Input : Two trees T_1 and T_2

Output : The Minimum cost sequence of
(1) merges, prunes
(2) contracts and
(3) relabelings
that transform the two input trees into a common tree.

Restriction on contract (related to vision application):
An edge can be contracted only if both its endpoints have degree greater than or equal to three.

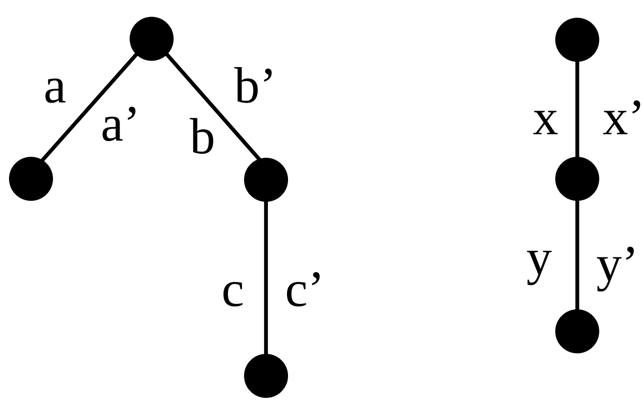
Basic Tree Edit distance

Earlier work : Zhang and Shasha's algorithm for tree edit distance with edit operations set = { edge contract, edge relabeling}.

Basic idea : *Dynamic Programming*

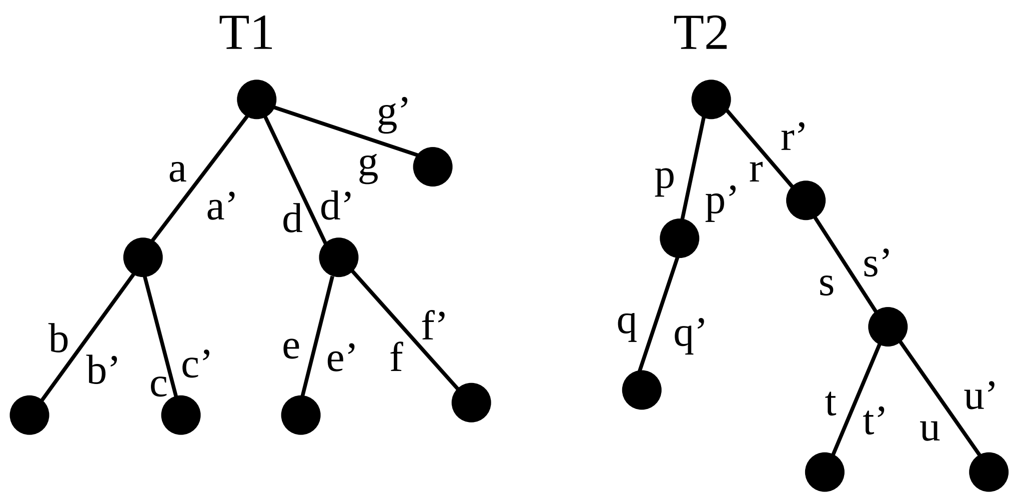
We give here our version of their algorithm, which is based on *Euler strings*.

Euler String:

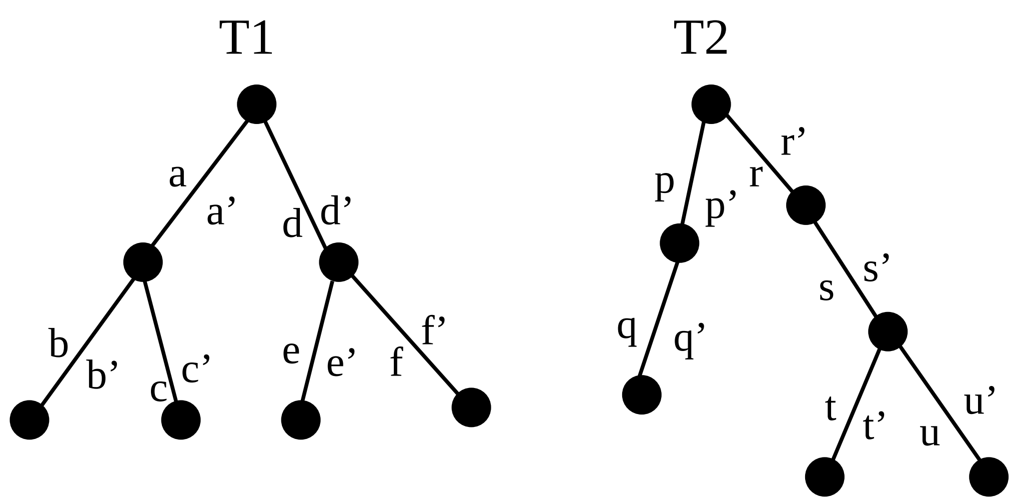


Left : $aa'bcc'b'$ Right : $xyy'x'$

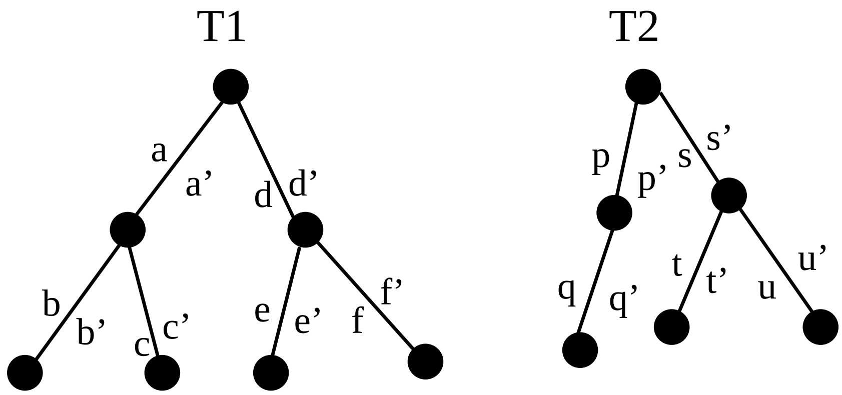
Let's look at an edit sequence on a pair of trees and see how it affects the corresponding euler strings.



Original Pair of trees.
$(a \ldots g', p \ldots r')$.

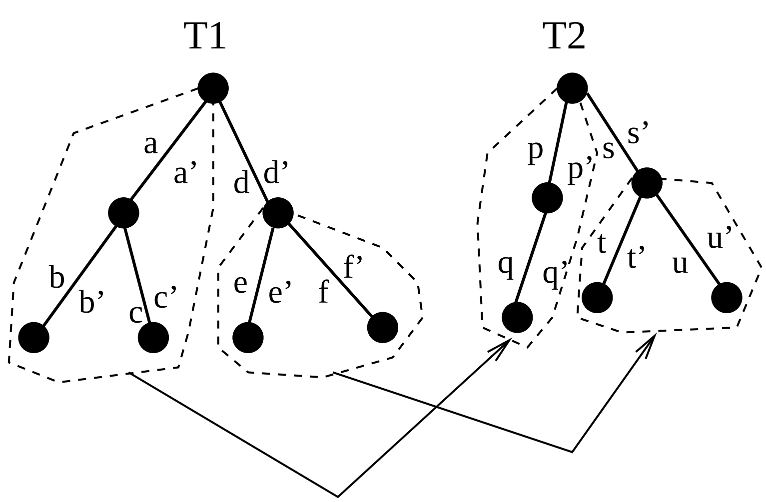


After contract g' on left
$(a \ldots d', p \ldots r')$.



Contract r' on right $(a \ldots d', pqq'p'stt'uu's')$.
T2's euler string can also be written as
$(pqq'p'$ **r** $stt'uu's')$, so that

(1) it is a substring of $p \ldots r'$.
(2) it still does not contain r (dart r' is absent)



Match d' to s'. This leads to two subproblems:
(1) $(a \ldots a', p \ldots p')$.
(2) $(e \ldots f', t \ldots u')$.

Algorithm for basic tree edit distance

- Dynamic programming on set of all possible pairs of Euler strings of T_1 and T_2.

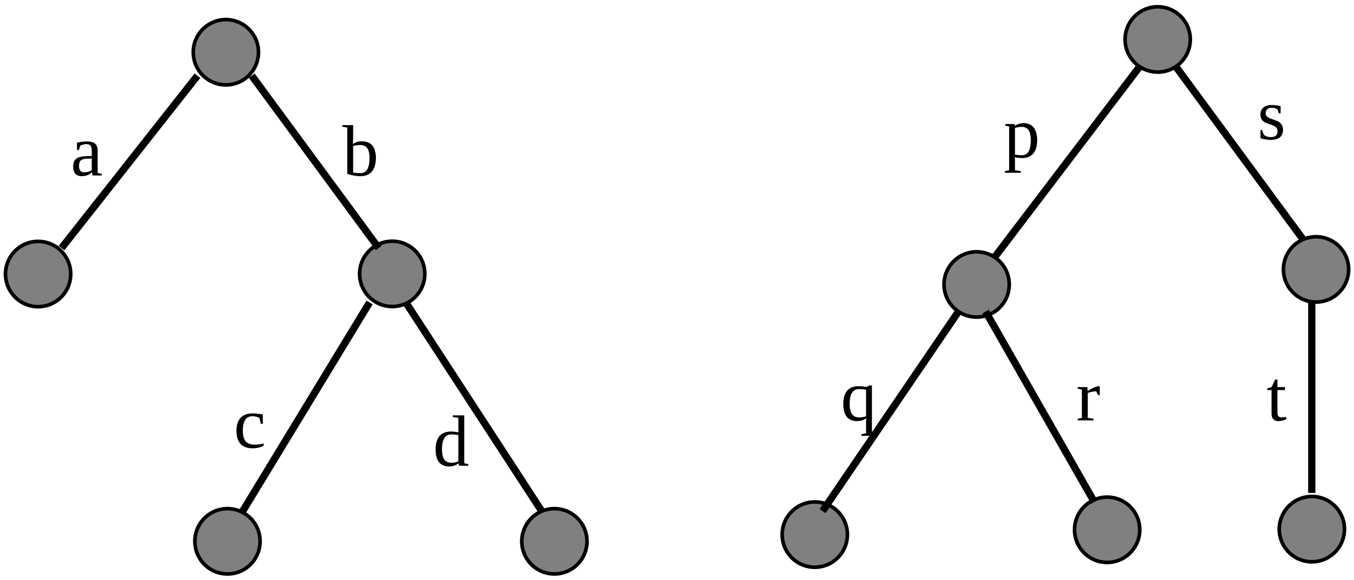
- A subproblem is (s_1, s_2) where s_i is a substring of the Euler string of T_i.

- Cost of a subproblem can be computed from the costs of a few other "smaller" subproblems, shown overleaf.

- What could happen to the rightmost edges (e_1, e_2) of the Euler strings in the optimal edit sequence?

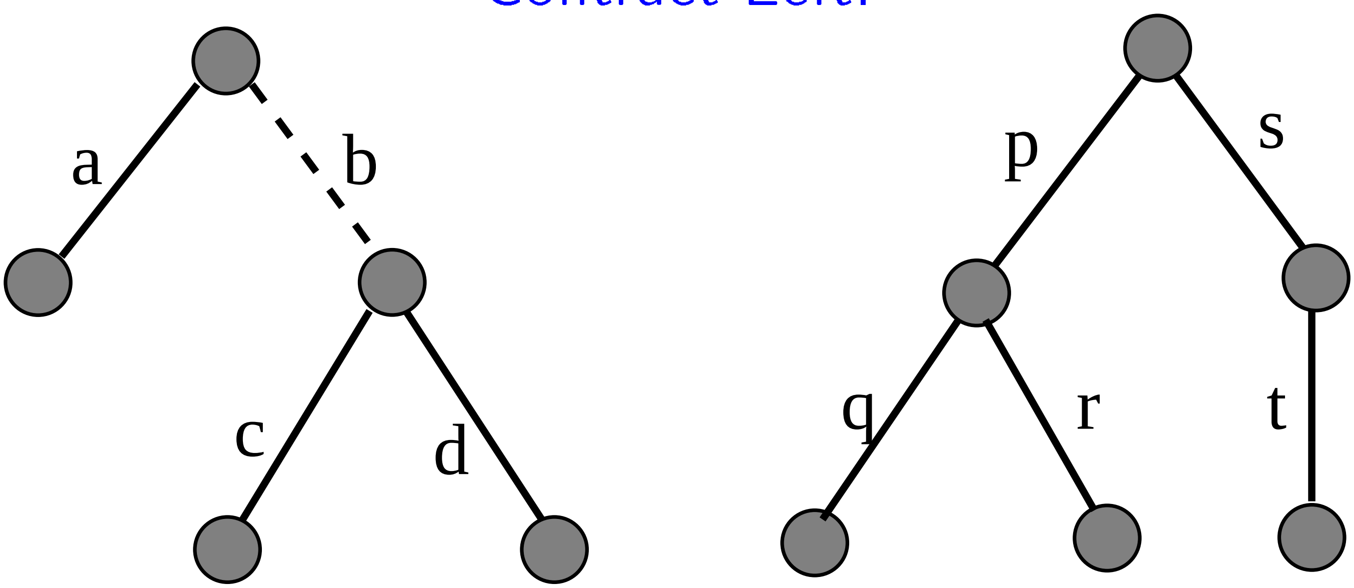
  (1) e_1 gets contracted
  (2) e_2 gets contracted
  (3) e_1 gets matched to e_2

Initial Subproblem:



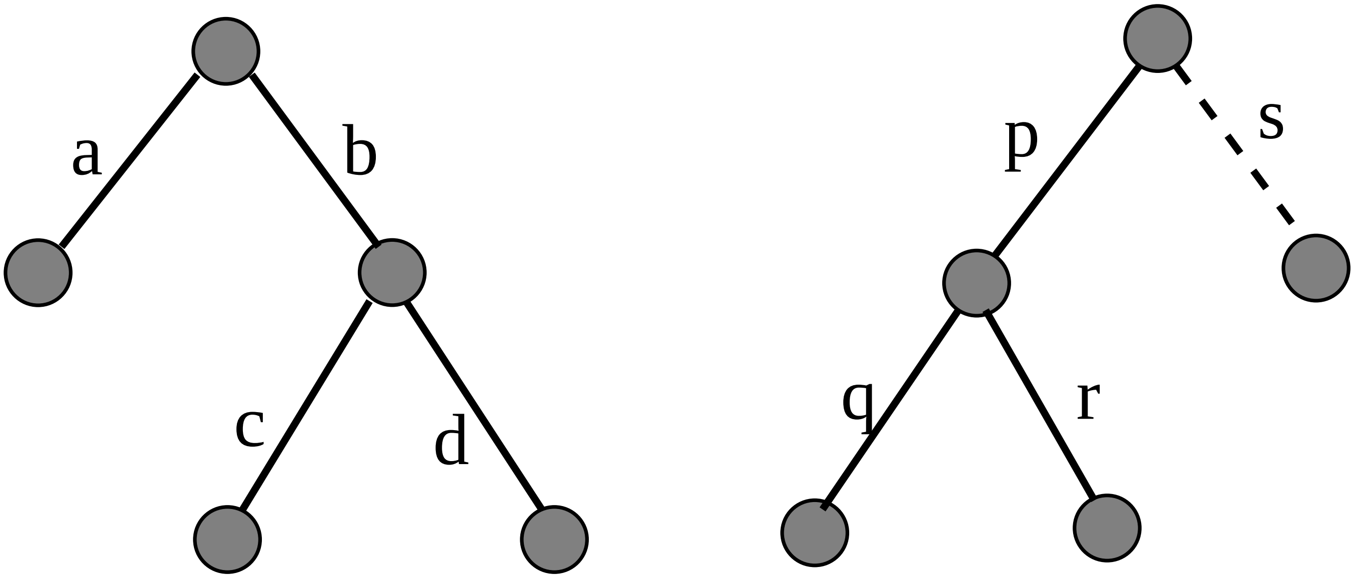
$(aa'bcc'dd'b', pqq'rr'p'stt's')$

Contract Left:



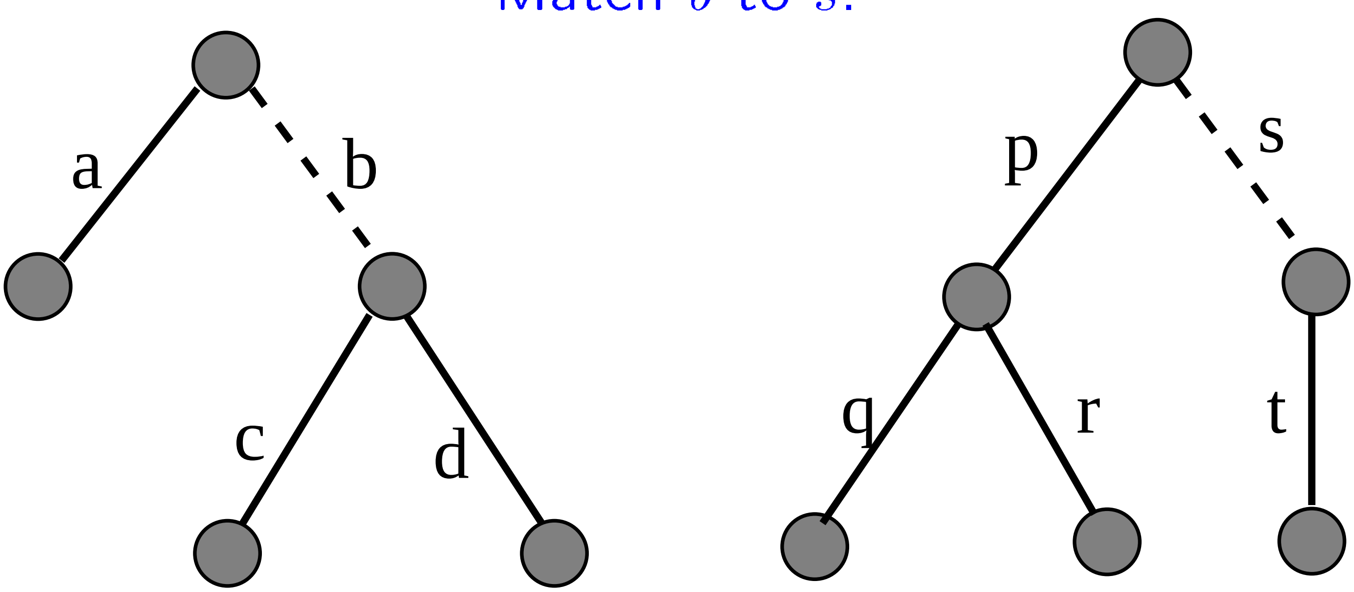
$(aa'bcc'dd', pqq'rr'p'stt's')$

Contract Right:



$(aa'bcc'dd'b', pqq'rr'p'stt')$

Match b to s:

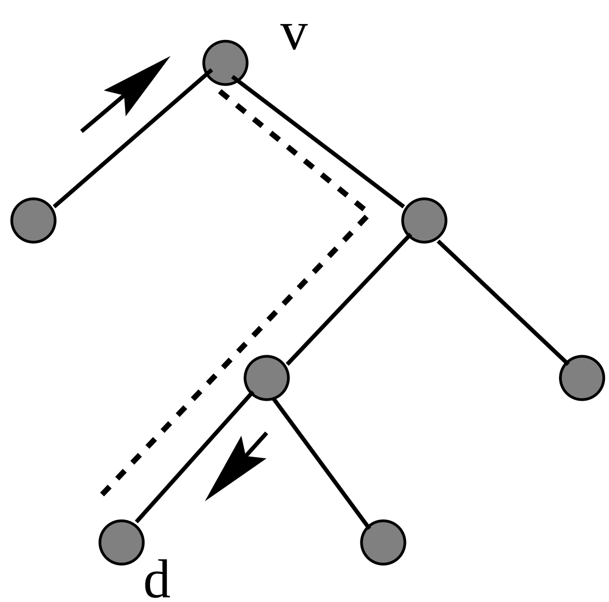


$(aa', pqq'rr'p')$ and $(cc'dd', tt')$

Algorithm with merge and prune

- Each half of our subproblems has just one merged edge.
- formed by merging consecutive edges on root-leaf path
- Represent it using one extra parameter, v in each half of the subproblem.

We merge a path into a single edge. v is the top of the path.



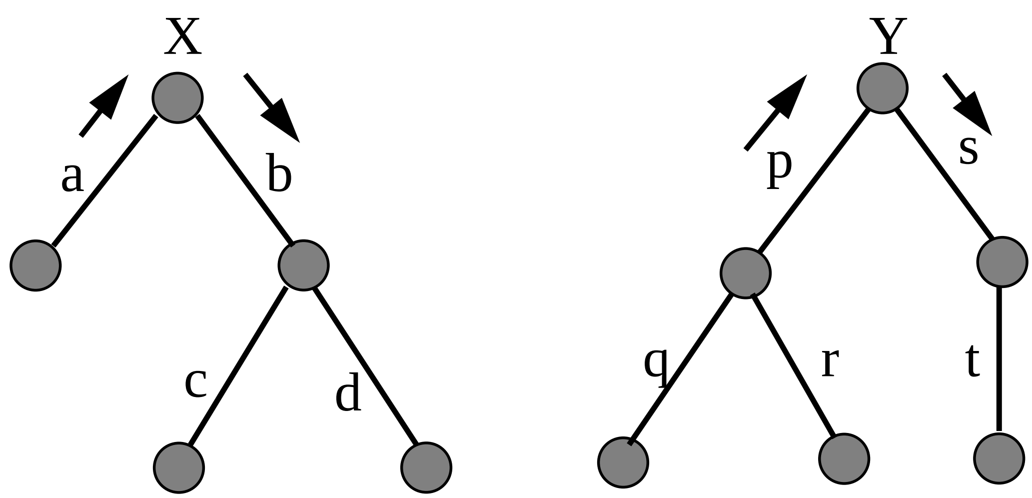
The arrows show the start and end of the Euler string. The path from d to v (dotted lines) is merged into a single edge.

Algorithm for merge and prune - continued

Again look at all possible operations on the rightmost edges.

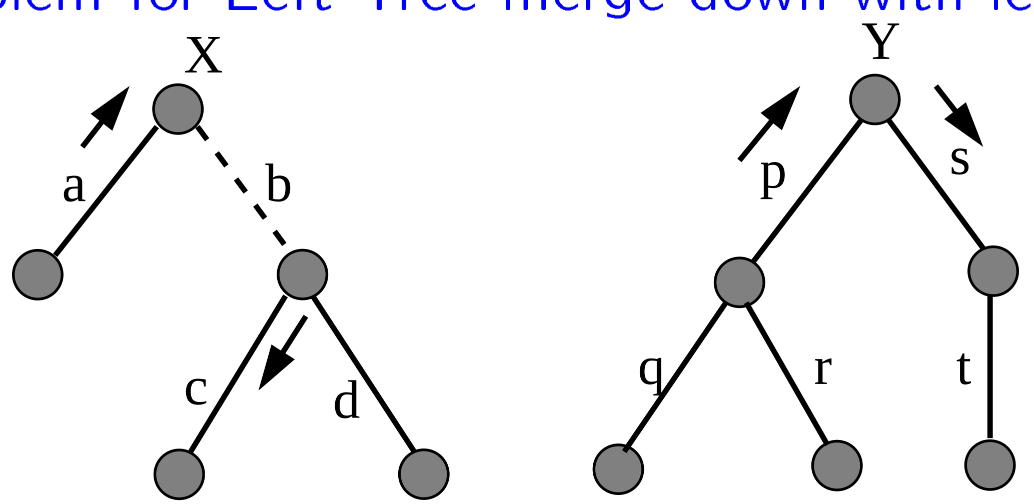
- contract the merged edge in T_1
- contract the merged edge in T_2
- further merge the edge in T_1 with its descendant
- further merge the edge in T_2 with its descendant
- match the two merged edges

Initial Subproblem - Arrows denote the start and endpoints of the Euler string. X and Y are the top of the merged edges.



$$(aa'bcc'dd'b', X, pqq'rr'p'stt's', Y)$$

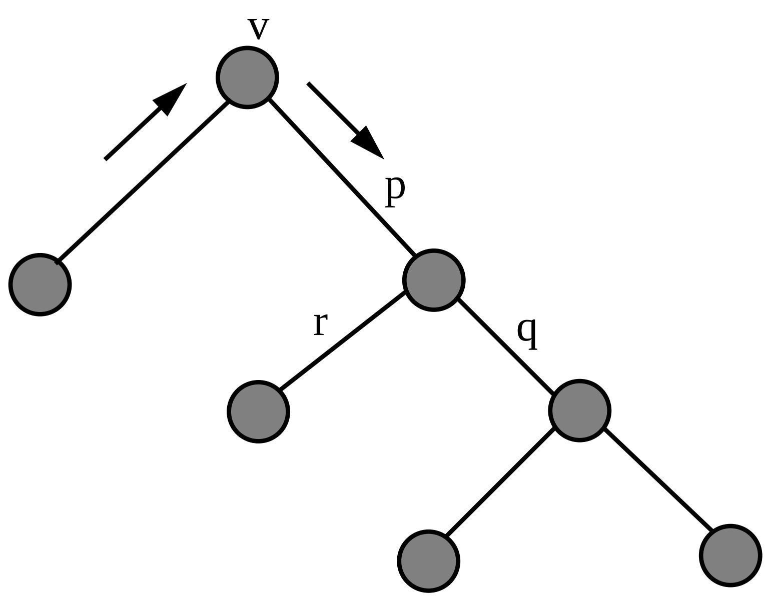
Subproblem for Left Tree merge down with left child.



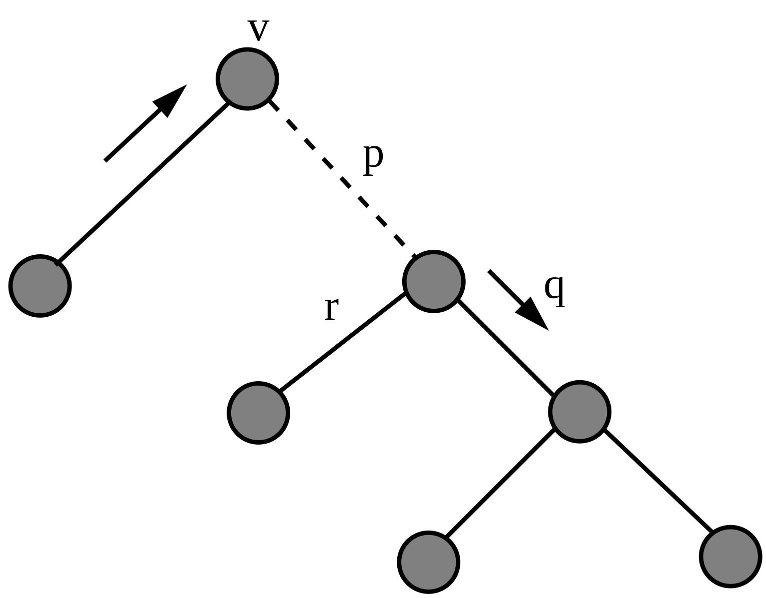
$$(aa'bcc', X, pqq'rr'p'stt's', Y)$$

Problem with pruning away left child

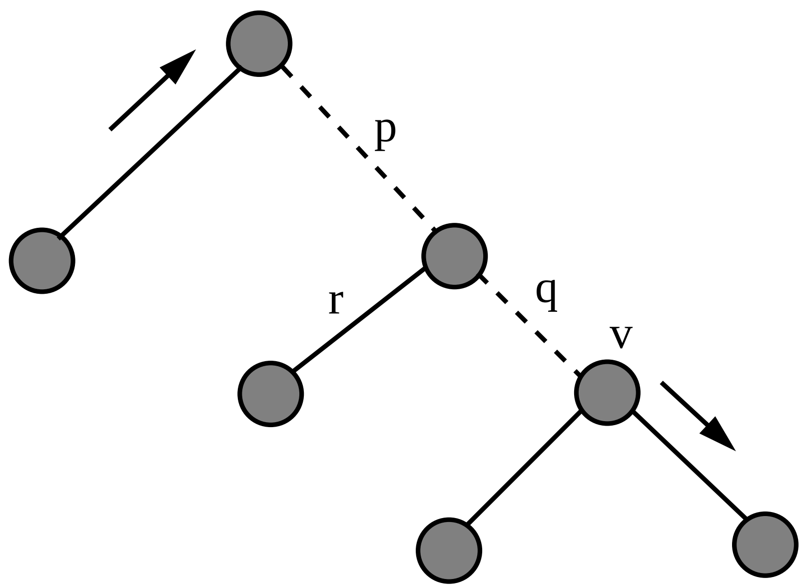
The Initial Half Subproblem



After merging p with q (pruning our r)



After contracting q



In this subproblem it's now ambiguous whether r was pruned out or not.

The same sub-problem could have been reached by contracting p and q separately!

The Fix

If you want to merge a few edges and then contract the result, then do it in two parts.

(1) contract p
(2) contract q
(3) when you encounter r, decide whether to prune it off or not.

This imposes a condition on the cost function (refer to paper).

Complexity

Time Complexity : number of subproblems
$O(n_1^2 n_2^2 d_1 d_2)$,
where n_i is size of T_i and
d_i is the depth of T_i.

Space complexity : We don't have to store the solution
to every subproblem all the time.
$O(n_1 n_2)$.

Future work

- Empirical Evaluation of the algorithm
- Faster algorithms
- Extend to matching 3D surfaces