Efficient Parallel Computation of Nearest Neighbor Interchange Distances
(Preliminary Version)

Mikael Gast* Mathias Hauptmann†

The nni-distance is a well-known distance measure for phylogenetic trees. We construct an efficient parrallel approximation algorithm (in the CRCW-PRAM model) for the nni-distance. Given two phylogenetic trees T_1 and T_2 on the same set of taxa and with the same multi-set of edge-weights, the algorithm constructs a sequence of nni-operations of weight at most $O(\log n) \cdot \text{opt}$, where opt denotes the minimum weight of a sequence of nni-operations transforming T_1 into T_2. This algorithm is based on the sequential approximation algorithm for the nni-distance given by Das-Gupta et al. [DHJ+00].

1. Introduction

Phylogenetic trees (or *phylogenies*) are a well-known model for the history of evolution of species. Such a tree represents the lineage of a set of todays species, or more generally a set of *taxa*, which are located at the leaf-level of the tree. The set and the ordering of the internal nodes describe the ancestral history and interconnections among the taxa. Usually phylogenetic trees have internal nodes of degree 3. A *weighted* phylogeny additionally imposes *weights* on its edges, representing the evolutionary distance between two taxa. We call a phylogeny *unrooted* or *rooted*, for the latter case if a common eldest ancestor is known.

Concerning the reconstruction of phylogenetic trees from a given set of genetic data, a variety of different schemes and algorithms were introduced over the past decades. Each method is based on a different objective criterion or distance function in the course of construction — for example parsimony, compatibility, distance and maximum likelihood. Due to this, the resulting phylogenies may vary according the internal topology and leaf configuration, although they have been created over the same set of taxa. Hence it is a reasonable approach to compare different

*Dept. of Computer Science, University of Bonn. e-mail:gast@cs.uni-bonn.de
†Dept. of Computer Science, University of Bonn. e-mail:hauptman@cs.uni-bonn.de

phylogenies for their similarities and discrepancies. As well for this task many different measures were proposed, including subtree transfer metrics, minimum agreement subtrees et cetera.

In this paper we focus on a restricted subtree transfer measure to compare phylogenetic trees, i.e. the *nearest neighbor interchange distance* (nni), which was introduced by D.F. Robinson in 1971 [Rob71]. A *nni-move* operationally swaps two subtrees, which are immediately connected but separated by an edge in the tree. Consequently, the *nni-distance* between two trees is the minimal number of restricted nni-operations required to transform one tree into the other. See Figure 1 for an illustration of the nni-operation.

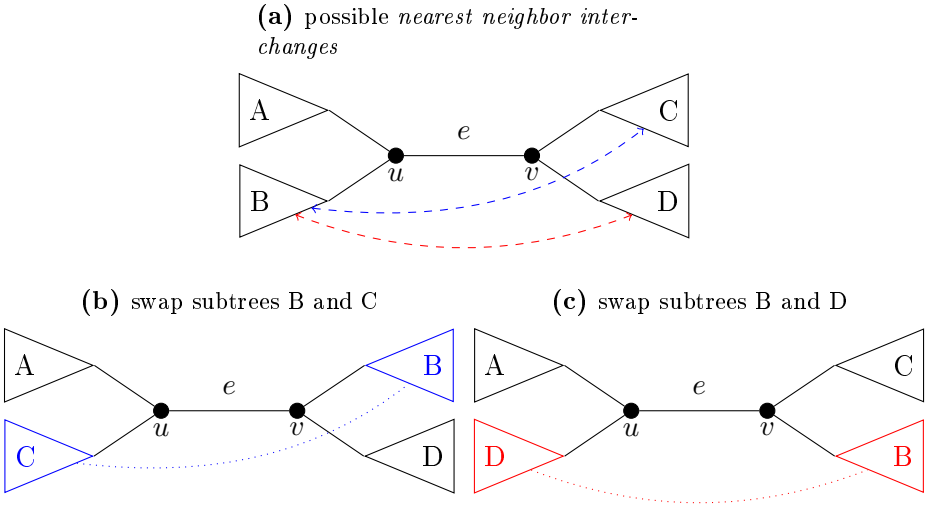

Figure 1: The possible nni-moves relative to an internal edge $e = (u, v)$. Each triangle A,B,C,D represents a subtree of the tree. The uniform cost of this operation is $wt(e)$.

Next we want to summarize some important results concerning the nni-distance measure.

1.1. Previous Results

Although the nni-distance has a transparent definition in terms of a simple transformation of trees, the efficient and fast computation indeed has showed to be surprisingly challenging.

For more than a decade, since its introduction in 1971 by D.F. Robinson [Rob71], no computationally efficient algorithm for measuring the nni-distance was known for practically large instances of phylogenetic trees. William H.E. Day and Edward K. Brown were the first to present an efficient approximation algorithm for unweighted instances in 1985 [Day85]. For unweighted phylogenies it required $\mathcal{O}(n \log n)$ time for unrooted and $\mathcal{O}(n^2 \log n)$ time for rooted instances. But it remained unclear whether the computational problem was *NP-hard* or not.

In 1996 Li, Tromp and Zhang considered the maximum nni-distance between arbitrary 3-regular trees [LTZ96] and gave logarithmic lower and upper bounds on $\Delta(G)$ (the collection graph of 3-trees with edges denoting that two trees are one nniapart). Furthermore they disproved some faulty results regarding the *decomposability property* and NP-hardness of the nni-distance and gave a new approach regarding these topics. In the last sections of their paper they formed

a result on the *approximation ratio* (or A.R. for short), which is $\log n + \mathcal{O}(1)$ for polytime approximation algorithms on unweighted instances.

Later on, in 1997, Bhaskar DasGupta, Xin He and Tao Jiang joined the group around Li whilst a visiting stay of some authors at Waterloo University. Together they achieved not only to prove the NP-completeness of computing the nni-distance on weighted and unweighted instances, but also the same result on trees with unlabeled (or non-uniformly labeled) leaves. As an algorithmic result they gave an $\mathcal{O}(n^2)$ approximation algorithm with A.R. $4\log n + 4$ for weighted instances [DHJ+97], which will be the foundation and guide-line of our work regarding the efficient parallel computation of the nni-distance. Moreover, they observed that the nni-distance is identical to the *linear-cost subtree-transfer* distance on unweighted phylogenies [DHJ+99] and that an exact algorithm for distance-restricted instances can be found with running time $\mathcal{O}(n^2 \log n + n \cdot 2^{11d})$.

Finally they have formulated their results again and more elaborately in 2000, appearing as a paper in [DHJ+00].

1.2. Our Work

Based on the above mentioned approximation algorithm by DasGupta, operating in time complexity $\mathcal{O}(n^2)$ and with A.R. $4(1 + \log n)$, we present an efficient parallel approximation scene for computing the nni-distance on weighted phylogenies. In order to do that, we formulate pseudo-code algorithms with polylog time complexity asserting a polynomial number of processors/processes, and thereby show, that each step of DasGupta's algorithm is in *Nick's Class* (*NC* for short) in terms of parallel computation.

The rest of the paper is organized as follows: In Chapter 2 we formulate some preliminaries in graph theory, phylogeny and computational complexity. Chapter 3 will give a short description of DasGuptas approximation algorithm. Then followed by Chapter 4 in which our efficient parallel approximation scene is presented at some detail.

2. Preliminaries

In this section we give the formal definition of phylogenies, nearest-neighbor interchange operations and the nni-distance measure.

We will make use of the following notation. Let $T = (V, E)$ be an undirected or directed tree, then $\mathcal{L}_T \subseteq V$ denotes the set of *leaves* of T and $\mathcal{I}_T \subseteq V$ the set of *internal vertices* of T.

The most important primitives in phylogenetic analysis are taxa and phylogenies.

Definition 2.1. *Given a finite set of* taxa $S = \{s_1, \ldots, s_n\}$, *a* phylogeny *for S is a triplet $T = (V, E, \lambda)$ where (V, E) is an undirected tree $\lambda : \mathcal{L}_T \to S$ is a bijection and such that every internal node of T has degree* 3. *A* rooted phylogeny *for S is a tuple $T = (V, E, \lambda, r)$ such that (V, E, λ) is a phylogeny and $r \in V$ is a distinguished node called the root of T. A* weighted phylogeny *for S is a tuple $T = (V, E, \lambda, w)$ such that (V, E, λ) is a phylogeny and $w : E \to \mathbb{R}^+$ is a weight function on the set of edges of T. A* rooted weighted phylogeny *is*

a tuple $T = (V, E, \lambda, w, r)$ *such that* (V, E, λ, r) *is a rooted phylogeny and* $w : E \to \mathbb{R}^+$ *is an edge-weight function.*

In order to compare different phylogenies on the same set of taxa (e.g. generated by different construction methods), one usually imposes a distance measure on the space of all phylogenies for the given set of taxa.

Various distance measures for comparing phylogenies have been investigated in the literature. The nni-distance measures the minimum number of nearest neighbor interchanges (nni) needed in order to transform one tree into another [RF79].

Definition 2.2. *Let* T *be a phylogeny (possibly rooted and/or weighted) and let* e_1, e_2, e_3 *be three edges of* T *that build a path of length three in* T *(in this order). The associated* nni-operation*, denoted as a triplet* (e_1, e_2, e_3)*, transforms the tree* T *into a new tree* T' *by swapping the two subtrees below the edges* e_1 *and* e_3 *as shown in the Figure 2. In this configuration we call the center edge* e_2 *the* operating edge*. In case of weighted phylogenies the* cost *of this nni-operation is defined as* $w(e_2)$*.*

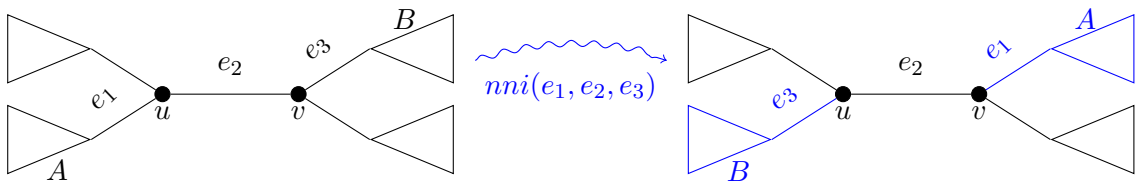

Figure 2: The nni-operation on T of the subtrees A and B defined by the triplet (e_1, e_2, e_3).

The associated genetic distance measure is the *nni-distance*:

Definition 2.3. *Let* S *be a set of taxa and let* T_1, T_2 *be phylogenies for* S*. The* nni-distance $d_{nni}(T_1, T_2)$ *of* T_1, T_2 *is defined as the* minimum length *of a sequence of nni-operations that transforms* T_1 *into* T_2 *(and* ∞ *in case no such sequence exists). In case of weighted phylogenies* $d_{nni}(T_1, T_2)$ *is the* minimum cost *of a sequence of nni-operations that transforms* T_1 *into* T_2*.*

Given two weighted phylogenetic trees $T_i = (V, E, \lambda, w_i)$, $i = 1, 2$ for the same set of taxa S, the following two conditions are necessary for the two trees to have a finite nni-distance.

1. For each taxon $s \in S$, let $e_i(s) \in E_{T_i}$ be the edge incident to the leaf with label s in T_i $(i = 1, 2)$. Then $e_1(s)$ and $e_2(s)$ must have the same edge weight: $w_1(e_1(s)) = w_2(e_2(s))$.
2. $M_{T_1} = M_{T_2}$, where M_{T_i} denotes the multiset of edge-weights of T_i.

In order to identify parts or subtrees of the tree that require a "large" or "small" amount of work to be transformed into their counterparts from the other tree, the notion of *good edge-pairs* and *bad edges* or *non-shared edges* according to the set of leaf-labels and edge-weights is used in the literature (cf. [RF79, DHJ+00]).

Definition 2.4. (Good Edge Pairs, Bad Edges)
Let T_1 and T_2 be two weighted phylogenies for the set of taxa S. Two internal edges $e_i \in E_{T_1}$ and $e_j \in E_{T_2}$ form a good *edge-pair iff the following conditions hold:*

1. $w_1(e_i) = w_2(e_j)$.
2. *Both edges induce the same partition of the multiset of edge-weights on T_1 and T_2.*
3. *Both edges induce the same partition of the set of leaf-labels on T_1 and T_2.*

An edge $e_i \in E_{T_1}$ is called bad *if there does not exist any edge $e_j \in E_{T_2}$ such that (e_i, e_j) forms a good edge-pair.*

If e_i and e_j form a good edge pair, no nni-move with operating edge e_i is needed to transform T_1 into T_2.

3. DasGupta's Sequential Approximation Algorithm

In this section we give an outline of DasGupta's approximation algorithm [DHJ+00] for the nni-distance on weighted phylogenies on a set S of n taxa. For the ease of notation we also presume/conjecture that all considered phylogenies are rooted at the same arbitrary leaf r. Unless otherwise mentioned we will reference to these rooted and weighted phylogenies on S with the term *phylogeny* for short.

Theorem 3.1. [DHJ+00] *Let T_1 and T_2 be two phylogenies. Then $d_{nni}(T_1, T_2)$ can be approximated within $\mathcal{O}(n^2)$ time and A.R. $4(1 + \log n)$.*

Given two phylogenies T_1, T_2, at first the multisets of edge-weights of internal edges of both, T_1 and T_2, are sorted in $\mathcal{O}(n \log n)$ time. In case these two multisets differ, T_1 and T_2 do not have a finite nni-distance. Hence, from now on we assume that $\{w_1, w_2, \ldots, w_{n-3}\}$ is the multiset of edge-weights of internal edges of both T_1 and T_2 and that $w_1 \leq w_2 \leq \cdots \leq w_{n-3}$ holds. Furthermore let $W := \sum_{i=1}^{n-3} w_i$ be the sum of all edge weights of internal edges of $T_i (i = 1, 2)$.

The following lemma provides a lower bound on $d_{nni}(T_1, T_2)$ in terms of W and the existence of good edge-pairs.

Lemma 3.1. [DHJ+00] *Assume that $d_{nni}(T_1, T_2) < \infty$. If T_1 and T_2 have no good edge pairs, then $d_{nni}(T_1, T_2) \geq W_{T_1} = W_{T_2}$.*

DasGupta's algorithm makes use of two different trees associated to each of the given phylogenies T_1, T_2, which we call the *auxiliary tree* and the *linear tree*.

Let $T = (V, E, \lambda, w, r)$ be a phylogeny. An *auxiliary tree* $A_T = (V, E', \lambda, w', r)$ is a phylogeny on the same set of vertices V and labeling of taxa λ that has the following properties:

- all leaves $l, l' \in \mathcal{L}_{A_T}$ are of balanced height, $|depth_{A_T}(l) - depth_{A_T}(l')| = 1$,
- the multisets of edge-weights in the trees T and A_T are the same, $M_T = M_{A_T}$,

- the edge-weights of internal edges on every path from r to a leaf in A_T are non-descending.

Having the set M_T of edge-weights sorted, such that $w_1 \leq w_2 \leq \cdots \leq w_{n-3}$ holds, we achieve the auxiliary tree property while arranging the edge-weights in M_T on an binary balanced tree such that, at level i, w_{2^i-1+j} is the j-th edge-weight assigned to an edge from the left. In a first step DasGupta's algorithm constructs an auxiliary tree $A_{T_i} = (V_i, E_i', \lambda_i, w_i', r_i)$, $i = 1, 2$, for both T_1 and T_2.

In a second step both the original phylogenies T_i and the associated auxiliary trees A_{T_i} are transformed into so called *linear trees*: For a given phylogeny $T = (V, E, \lambda, w, r)$, a *linear tree* $L_T = (V, E'', \lambda, w'', r)$ of T is a phylogeny over the same labeling λ and such that every internal node is adjacent to at least one leaf (see Figure 3 for an example).

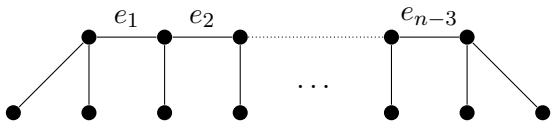

Figure 3: The linear tree L with internal edges $e_1, e_2, \ldots, e_{n-3}$.

Now, a short analysis of DasGupta's algorithm together with a pseudo-code description is presented. First for the special case that every edge $e_i \in T_1$ is *bad* and, by Lemma 3.1 on the previous page, with a total cost of at most $(4 + 4 \log n)W$.

Description of DasGupta's algorithm:

1. Choose an arbitrary leaf r as root and transform T_1 into a balanced binary tree T_1' of height $\lceil \log n \rceil$. The internal nodes are put in place such that any path from the root to a leaf has non-decreasing edge-weights. Therefore at the ith level ($i \geq 1$), e_{2^i-1+j} ($0 \leq j < 2^i$) is the jth edge from the left. The transformation of Step 1 consists of three phases:

   1a. Transform T_1 to a *linear* tree L such that the edges $e_1, \ldots, e_{n-3}$ appear in an arbitrary order from left to right. In order to do that, form a *left path* P starting at a leaf r. If P contains all internal edges, T_1 has been transformed into a linear tree L_1. At most one nni-move is performed on each internal edge of T_1, thus Phase 1.1 costs at most W and can be completed in $\mathcal{O}(n)$ time. An example of a linear tree is depicted in Figure 3.

   1b. Similar to Phase 1a, transform T_1' into a linear tree L' in $\mathcal{O}(n)$ time and with cost W. The internal edges in L' appear as $e_1'', \ldots, e_{n-3}''$.

   1c. Use an analogue variant of *merge sort* to transform L to L', performed in $\mathcal{O}(n \log n)$ time and with costs $W \log n$.

   In order to achieve the transformation of T_1 to T_1', perform the nni-moves of Phase 1a, followed by the nni-moves of Phase 1c, followed by the *inverse* of the nni-moves of Phase 1b. In total Step 1 can be completed in $\mathcal{O}(n \log n)$ time and cost at most $(2 + \log n)W$.

2. Analogue to Step 1 on the preceding page transform T_2 to T_2' in $\mathcal{O}(n \log n)$ time and with cost $(2 + \log n)W$, and note that internal structure of T_2' equals the structure of T_1'.

3. Transform T_1' to T_2' in $\mathcal{O}(n \log n)$ time and with cost $2(\log n)W$.

   To finally transform T_1 to T_2, perform the nni-moves of Step 1 on the previous page, followed by the nni-moves of Step 3, followed by the *inverse* nni-moves of Step 2. So the algorithm can be completed in $\mathcal{O}(n \log n)$ time and total costs of $4(1 + \log n)W$.

Algorithm 1: DasGupta's_Sequential_Algorithm

Input: Rooted phylogenetic trees T_1, T_2.
Output: nni-distance $d_{nni}(T_1, T_2)$ and a sequence $\mathcal{N}$ of nni-operations transforming T_1 into T_2.

begin
  for $i = 1, 2$ **do**
1 Construct *auxiliary trees* A_{T_i};
   `/* generate nni-sequence` $\mathcal{N}_i$ `to transform` T_i `into` A_{T_i} `*/`
2 Generate sequence $(t_{i,1}, \ldots, t_{i,j(i)})$ that transforms T_i into a *linear tree* L_{T_i};
   Generate sequence $(a_{i,1}, \ldots, a_{i,k(i)})$ that transforms A_{T_i} into a *linear tree* $L_{A_{T_i}}$;
3 Generate *merge-sort*-sequence $(s_{i,1}, \ldots, s_{i,l(i)})$ that transforms L_{T_i} into $L_{A_{T_i}}$;
   $\mathcal{N}_i := (t_{i,1}, \ldots, t_{i,j(i)}, s_{i,1}, \ldots, s_{i,l(i)}, ; a_{i,k(i)}, \ldots, a_{i,1})$;
   `/* note that sequence` $(a_{i,1}, \ldots, a_{i,k(i)})$ `is reversed in order to allow back-transformation to` A_{T_i} `*/`
4 Generate sequence $(b_1, \ldots, b_m)$ to transform A_{T_1} into A_{T_2};
   $\mathcal{N} := \mathcal{N}_1 \circ (b_1, \ldots, b_m) \circ \mathcal{N}_2'$;
   `/* note that sequence` $\mathcal{N}_2'$ `is reversed in order to allow back-transformation to` T_2 `*/`
end

Next we consider the case that T_1 and T_2 have some *good* edge pairs. Here we have to identify the set $E' \in E_{T_1}$ of edges in T_1 that form good edge pairs with edges in T_2 (see [DHJ+00] for details). Then, every edge in E' induces a subtree in T_1 consisting of one or more *connected components* each of which is a subtree of T_1. These connected components with total weight W' can be found in $\mathcal{O}(n)$ time. To finally transform T_1 to T_2 we perform the algorithm stated above on each such component. The algorithm takes $\mathcal{O}(n^2)$ time, the total cost is bounded by $4(1 + \log n)W'$. This completes the proof of the main Theorem 3.1 on page 5 of this section.

4. Parallel Computation of the nni-Distance

In this section we construct efficient parallel algorithms for the three steps of DasGupta's algorithm (in the CRCW-PRAM-model).

When T is a 3-regular phylogeny (i.e each internal node has degree 3 in T), the internal nodes of T can be classified with respect to the number of adjacent leaves.

Definition 4.1. *Let $T = (V, E, \lambda, w)$ be a 3-regular phylogeny (each internal node has degree 3 in T). Let $\mathcal{L}$ be the set of leaves in T. An internal node $v \in \mathcal{I} = (V \setminus \mathcal{L})$ is called*

- *an* endnode *($v \in V_{end}$), if it is adjacent to two leaves and one internal node,*
- *a* pathnode *($v \in V_{path}$), if it is adjacent to one leaf and two internal nodes,*
- *a* junction-node *($v \in V_{junc}$), if it is adjacent to three internal nodes in T.*

This notation will be used in the course of the *linearization* of phylogenetic trees.

4.1. Detecting Good Edge-Pairs

We give two different parallel algorithms for the detection of good edge-pairs. The first one is based on the efficient parallel computation of connected components, the second one uses a bottom-up subtree pruning strategy.

Good Edge-Pairs via Connected Components Given the two trees T_1, T_2, the first algorithm (Algorithm 4 on page 10) considers all pairs (e_x, e_y) of edges e_x from T_1 and e_y from T_2 with $w(e_x) = w(e_y)$ in parallel. For each such pair, Algorithm 4 on page 10 computes in parallel the connected components of $T_1 \setminus e_x$ and $T_2 \setminus e_y$, generates the associated edge- and leaf-partitions and – based on this data – decides if e_x, e_y is a good edge-pair.

Detecting Connected Components There are well-known efficient parallel algorithms for the computation of the connected components of a given graph. One of the first such algorithms is due to D.S. Hirschberg [Hir76]. Here we make use of the algorithm of Goddard et al. [GKP94] which also works efficiently on mesh-like parallel systems (Algorithm 2 on the next page).

Given $T^e = (V, E \setminus \{e\})$, this algorithm computes a labeling $c_{T^e} : V \to \{0, 1\}$. The running time of this algorithm for trees is $\mathcal{O}(\log n)$ on $2n - 2$ processors. Figure 4 illustrates how T topologically falls into two partitions if edge e is removed from the tree.

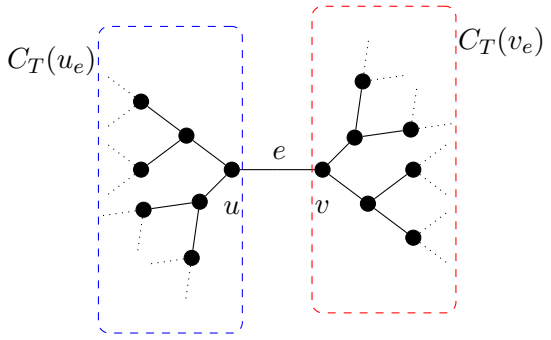

Figure 4: Components $C^u_{T^e}, C^v_{T^e}$ induced by edge $e = (u, v)$ in T.

Algorithm 2: PARALLEL_CONNECTED_COMPONENTS

Input: A phylogeny T with vertices uniquely labeled in $\{0, 1, \ldots, 2n-2\}$.
Output: The minimal component number $c(v)$, $\forall v \in V$.
begin
  foreach $v \in V$ **parallel do** /* initialize pointers */
    $c(v) := \min\{v, \min\{u|$ vertex u is adjacent to v in $T\}\}$;
  repeat
    foreach $v \in V$ **parallel do** /* opportunistic pointer jumping */
      $c_{old}(v) := c(v)$;
      $c'(v) := c(\min\{c(v), \min\{c(u)|$ vertex u is adjacent to v in $T\}\})$;
    foreach $v \in V$ **parallel do** /* tree hanging */
      $c(v) := \min\{c'(v), \min\{c'(u)|c(u) = v\}\}$;
    foreach $v \in V$ **parallel do** /* normal pointer jumping */
      $c(v) := c(c(v))$;
  until $c = c_{old}$;
end

Generation of Edge- and Leaf-Partitions For a given tree T, edge e in T and the associated function $c_{T^e} : V \to \{0, 1\}$ we can compute the induced partitions of the set of taxa and of the multiset of edge-weights efficiently in parallel. More precisely the following procedure computes a partition $\gamma_{T^e} : S \to \{0, 1\}$ of the taxa and two multisets $W^0_{T^e}, W^1_{T^e} : \{w(e)|e \in E\} \to \mathbb{N}_0$ such that $W^j_{T^e}(w) =$ number of occurrences of weight w in the component j of T^e. These values are provided by algorithm 3.

Algorithm 3: EDGE_LEAF_PARTITIONS

Input: A phylogeny T and component numbers $c_{T^e}(v)$ for all $v \in V$ in T^e.
Output: The partition $\gamma_{T^e} : S \to \{0, 1\}$ of the taxa and two multisets $W^0_{T^e}, W^1_{T^e} : \{w(e)|e \in E\} \to \mathbb{N}_0$ such that $W^j_{T^e}(w) =$ number of occurrences of weight w in the component j of T^e.
begin
  foreach $s \in S$ **in parallel do**
    $\gamma_{T^e}(s) := c_{T^e}(\lambda^{-1}(s))$; /* assign leaf partition number for all $s \in S$ */
  for $j = 0, 1$ **do**
    $W^j_{T^e} := W$; /* initialize partitions with complete multiset */
  foreach $f \in E_{T^e}$ **in parallel do**
    for $j = 0, 1$ **do**
      if *f is in component j of T^e or $f = e$* **then**
        $W^{1-j}_{T^e}(w(f)) := W^{1-j}_{T^e}(w(f)) - 1$; /* reduce the manifold of weight $w(f)$ in counterpart partition */
end

Algorithm: Parallel Compute Good Edge-Pairs I Finally we obtain the parallel algorithm 4 for computing good edge-pairs.

Algorithm 4: GOOD_EDGE_PAIRS

Input: Phylogenies T_1, T_2.
Output: The set $\mathcal{G}_{T_1}^{T_2}$ of good edge-pairs (e_x, e_y) between T_1 and T_2 with $e_x \in E_{T_1}, e_y \in E_{T_2}$.

```
begin
    foreach e ∈ E_{T_i} in parallel do
        CONNECTED_COMPONENTS(T_i^e);                    /* returns c_{T_i^e}(v) */
        EDGE_LEAF_PARTITIONS(T_i^e, c_{T_i^e}(·));      /* returns γ_{T_i^e}(s), W_{T_i^e} */
    foreach e_x ∈ E_{T_1}, e_y ∈ E_{T_2} in parallel do
        if w(e_x) = w(e_y) then
            foreach s ∈ S in parallel do
                if γ_{T_1^{e_x}}(s) ≠ γ_{T_2^{e_y}}(s) then
                    break;
            foreach w ∈ {w(h)|h ∈ E} in parallel do
                if W_{T_1^{e_x}}(w) ≠ W_{T_2^{e_y}}(w) then
                    break;
            G_{T_1}^{T_2} = G_{T_1}^{T_2} ∪ {(e_x, e_y)};
end
```

Good Edge-Pairs via Bottom-up Propagation Alternatively one can compute the edge- and leaf-partitions in a bottom-up manner as follows: We choose some taxon $s \in S$ and replace T_i by the arborescense that results from T_i by choosing $r_i = \lambda_i^{-1}(s)$ as a root and orienting the edges of T_i appropriately.

Computing the orientations of T_1, T_2 Given an undirected tree T and a leaf $r \in \mathcal{L}_T$, one can efficiently generate the associated arborescense with root r by orienting the edges of T: We compute an Euler tour of T using the *Parallel Euler Tour* technique , splitting this tour at the root r and then applying the parallel prefix-sum algorithm.

Computing Edge- and Leaf-Partitions Given a rooted directed phylogeny T with root r we compute the lists $\mathcal{L}_T^v \subseteq S$ of leaf-labels and the multiset W_T^v of edge-weights in the subtrees T_v below v as shown in algorithm 5 on the next page.

Finally in Algorithm 6 on page 12 these subroutines are used in order to generate a list $\mathcal{G}$ of good edge-pairs efficiently in parallel.

Algorithm: Parallel Compute Good Edge-Pairs II After computing the edge-, leaf-partitions induced by every $v \in V_T$ and the corresponding subtree rooted at v, we are able to determine

Algorithm 5: SUBTREE_PARTITIONS

Input: Phylogeny T with root r and $parent(v)$ the parent of each node $v \in V_T$.
Output: For every internal node v the sets of leaf-labels $\mathcal{L}_T^v$ and multisets of edge-weights W_T^v of the subtree rooted at v.

```
begin
    foreach v ∈ V_T in parallel do
        if v ∈ L_T then
            L_T^v := {λ(v)};          /* initialize with leaf-label if v is a leaf */
            if v = r then
                W_T^v := {w(e)};      /* initialize with edge-weight of e = (v,u) with
                start point in v */
            else
                W_T^v := ∅;
        else
            L_T^v := ∅;
            W_T^v := {w(e)};                          /* initialize as above */
        next(v) := parent(v));                /* choose parent parent(v) as next */
        while next(v) ≠ r do
            L_T^next(v) := L_T^next(v) ∪ L_T^v;          /* propagate sets to next(v) */
            W_T^next(v) := W_T^next(v) ∪ W_T^v;
            next(v) := next(next(v));                        /* pointer-jumping */
end
```

the edge- and leaf-partition for every edge $e = (u, v)$, where u is pointing towards the root of the tree.

Observation 1. *$\mathcal{L}_T^v = \mathcal{L}_T^e$ and $M_T^v = M_T^e$ for edge $e = (v, u)$ and u is pointing towards the root of the tree.*

Finally we obtain the parallel algorithm 6 on the next page for alternatively computing good edge-pairs.

4.2. Linearizing Trees

Now both T_1, T_2 and their associated auxiliary trees T_1', T_2' are transformed into *linear trees* L_1, L_2 and L_1', L_2'. A *linear tree* is a phylogeny that only consists of *pathnodes* and *endnodes* (c.f. section 4 on page 7), i.e. every internal node is adjacent to at least one leaf (see also figure 5 on the next page).

Let us first give an outline of the linearization procedure which is divided into three phases:

1. *Activation-Phase:* We proceed in a bottom-up manner at the boundary of the tree, i.e. at endnodes $v \in V_{end}$ defined above. At every endnode v a process is started that builds the

Algorithm 6: GOOD_EDGE_PAIRS

Input: Phylogenies T_1, T_2.
Output: The set $\mathcal{G}_{T_1}^{T_2}$ of good edge-pairs (e_x, e_y) between T_1 and T_2 with $e_x \in E_{T_1}, e_y \in E_{T_2}$.

```
begin
    foreach T_i in parallel do
        PARALLEL_ROOTING_TREES(T_i, r);                    /* roots T_1, T_2 in r */
        SUBTREE_PARTITIONS(T_i);          /* returns L^v_{T_i} and W^v_{T_i} for all v ∈ V_{T_i} */
    foreach e_x ∈ E_{T_1}, e_y ∈ E_{T_2} in parallel do
        if w(e_x) = w(e_y) then
            foreach s ∈ S in parallel do
                if γ_{T_1^{e_x}}(s) ≠ γ_{T_2^{e_y}}(s) then
                    break;
            foreach w ∈ {w(h) | h ∈ E} in parallel do
                if W_{T_1^{e_x}}(w) ≠ W_{T_2^{e_y}}(w) then
                    break;
            G^{T_2}_{T_1} = G^{T_2}_{T_1} ∪ {(e_x, e_y)};
end
```

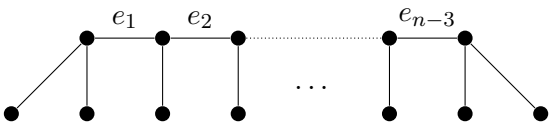

Figure 5: A linear tree L.

path to the next junction-node $u \in V_{junc}$ and *activates* u to prepare the junction node for insertion of the path from v.

If a junction-node u is activated by more than one endnode during the activation phase, among the two paths meeting at u we select the one of smaller weight for insertion. Let this path consist of k internal edges $e_1, \ldots, e_k$ where e_1 is incident to u.

2. *Insertion-Phase:* In the next phase, we generate the sequence of nni-operations that is used for the insertion of the selected path at the junction-node u. This yields a sequence of nni-operations of length k, the length of the path to be inserted. The internal edges $e_1, \ldots, e_k$ are the operating edges of these nni-moves.

3. *Update-Phase:* In the last phase the tree topology and the pointers inside the tree are updated after each Insertion-Phase.

These three phases are repeated until the tree T is transformed into a linear tree L.Since every iteration decreases the number of leaves by a factor of 2, the parallel running time of the linearization algorithm is bounded by $\lceil \log n \rceil$.

Lemma 4.1. *The number of iterations of phases 1-3 in the linearization algorithm is bounded by* $\lceil \log n \rceil$.

Proof. Let $|V_{end}| = k_0$ be the initial number of endnodes in T at the beginning of the first linearization-step. Now every endnode $v \in V_{end}$ tries to activate the next junction-node $next(v)$ towards the root of T. This will be successful for at least every second endnode, since one junction-node is shared by at most two endnodes. Therefore at least $\frac{k_0}{2}$ insertions of an endnode-path $path(v)$ is carried out at $next(v)$ in phase 1 of the linearization step. After phases 2 and 3 the number of end- and junction-nodes is reduced by at least $\frac{k_i}{2}$ in each step i. □

Orientate Edges towards Root

In order to determine the direction of insertion, we choose a new vertex or leaf as root r equally in both trees, if no root exists. Then, given C_u^e, C_v^e for every edge $e = (u, v)$, we generate an orientation $dir(e)$ on e pointing on either u or v if the respective endnode is in the same component with r. These values are provided by Algorithm 7.

Algorithm 7: ROOT_ORIENTED_EDGES

Input: Phylogeny T with root r and $c^e(v)$ the component numbers induced by e in T, $\forall e \in E(T), v \in V(T)$.

Output: For every edge e the root pointing node $dir(e) \in \{u, v\}$.

```
begin
    foreach e = (u, v) ∈ E(T) parallel do
        if c^e(r) = c^e(u) then                    /* test if root-component */
            dir(e) := u;                                 /* orientate edge e */
        else
            dir(e) := v;
end
```

Generate Endnode-Paths for Insertion

To describe the insertion process, Algorithm 8 on the next page provides for every node v the distance $dist(v)$, edge-list $path(v)$, length $length(v)$ and the head $head(v)$ of the path to the next junction- or endnode $next(v)$ heading towards root r. These values are computed efficiently via *parallel pointer jumping*.

Algorithm: Parallel Linearize Tree

We are now ready to formulate Algorithm 9 on page 15 for the linearization of a tree T. Figure 6 on the next page illustrates the notation given in the above algorithms and shows the result of an insertion-process.

Algorithm 8: To_Junction_Paths

Input: Phylogeny T with root r and pointer $parent(v)$ for all v in T and sets of junction- and endnodes V_{junc} and V_{end}.

Output: For every node v in T the values $dist(v)$, $path(v)$, $length(v)$, $next(v)$ and $head(v)$.

begin
  foreach $v \in V$ **parallel do**
    $dist(v) := w(e_v)$; `/* initialize with parent edge` $e_v = (v, parent(v))$ `*/`
    $path(v) := e_v$;
    $head(v) := v$;
    $length(v) := 1$;
    $next(v) := parent(v)$;
    while $next(v) \notin V_{junc} \cup V_{end}$ **do**
      $dist(v) := dist(v) + dist(next(v))$;
      $path(v) := path(v) \circ path(next(v))$;
      $head(v) := next(v)$;
      $length(v) := length(v) + length(next(v))$;
      $next(v) := next(next(v))$; `/* Pointer-Jumping */`
end

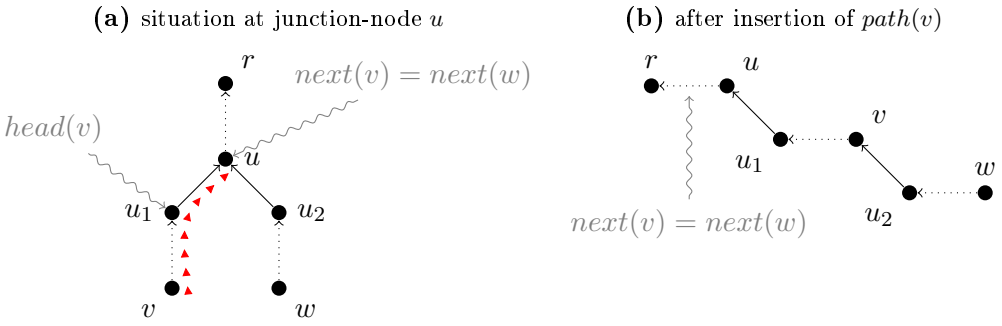

Figure 6: Insertion of $path(v)$ from endnode v adjoining junction-node $u = next(v)$.

We will utilize this linearization technique in Phase 1.1 and 1.2 of DasGupta's original algorithm. The superior aim of Phase 1 is to transform two phylogenies T_1 and T_2 into linear trees L_1 and L_2 with the same order on internal edges and then into balanced binary shapes T_1' and T_2' with the same internal topology. While the inverse nni-operations of Phase 1.2 transforms a linear tree into the balanced shape, in Phase 1.3 we have to use a parallel analogue of the sequential merge-sort stated by DasGupta to match the sequence of internal edges in both linear trees. This will be the focus of the upcoming section.

4.3. Sorting Linear Trees

After Phase 1.1 and 1.2 we now have two linear trees L_1 and L_1' associated to the original tree T_1 and the balanced tree T_1' with presorted edges. Now the sequence of nni-operations will be generated that transforms the sequence $e_1', e_2', \ldots, e_{n-3}'$ of internal edges in L_1 into the linearized

Algorithm 9: PARALLEL_LINEAR_TREE

Input: A phylogeny T with root r.
Output: A list $\mathcal{N}$of nni-operations for the transformation of T to L_T.

```
begin
    while ∃u ∈ V_junc do
        To_Junction_Paths(T);                    /* re-generate paths and pointers */
        foreach v ∈ V_end parallel do
            u := next(v);
            a(u) := v;                                        /* activate u from v */
        foreach active u ∈ V_junc parallel do
            u_1 := head(a(u));
            u_2 := (sib(u) ≠ head(a(u)));
            while i ≤ length(a(u)) parallel do
                nni(u) := nni(u) ∘ ((leaf(u_1), u_1), (u_1, u), (u, u_2));
                u_1 := sib(u_1);
                i := i + 1;              /* generate nni-triplets for the whole path */
            N := N ∘ nni(u);                              /* concatenate list of nni's */
            parent(u_2) := u_1;                          /* insertion of the path at u_2 */
            w((u_1, u_2)) := w((u_2, u));
            V_junc := V_junc \ {u};                                  /* deletion of u */
end
```

sorted sequence, say $e''_1, e''_2, \ldots, e''_{n-3}$, of L'_1.

The basic scheme of the sequential algorithm is, first, to transform adjacent edge-pairs by nni-moves, such that the whole sequence afterwards is pairwise alternating from ascending to descending according to the sorting order of $e''_1, e''_2, \ldots, e''_{n-3}$ (the ascending and descending subsequences of edges will be called *blocks*). Then, starting from the middle, we merge and pull out adjacent blocks, finally resulting in a linear tree of blocks of doubled size, again alternating. At k-th stage, we are starting with $\frac{n}{2^k}$ blocks of 2^k internal edges each, resulting in $\frac{n}{2 \cdot 2^k}$ blocks consisting of $2 \cdot 2^k$ edges. See Figure 7 on the following page for an example. If the resulting sequence consists only of one block, containing all edges, we are done with merge-sorting the linear tree.

4.3.1. Algorithm: Parallel Tree Merging

The next step is to fit the sequential algorithm into a parallel computation scheme. Therefore we may not only consider the two adjacent blocks in the middle for comparing and merging, but, all $\frac{n}{2^k}$ block-pairs that will be adjacent in the course of stage k in parallel. So we have to describe the pairing of blocks and edges inside blocks for each stage in order to allow parallel computation.

At stage k let $B_1, B_2, \ldots, B_{\frac{n}{2^k}}$ be the blocks appearing in that order on the linear tree. We start pairing recursively from the middle, such that B_l pairs with $B_{\frac{n}{2^k}-(l-1)}$ for $l \in \{1, \ldots, \frac{n}{2 \cdot 2^k}\}$.

Furthermore, let $e_{(l-1)2^k}, e_{(l-1)2^k+1}, \ldots, e_{l2^k}$ be the edges of block B_l at stage k. To preserve simplicity, we illustrate the merging of edges of two blocks within a pair (B_x, B_y), which is said to be the block-pair to get adjacent and to be merged at stage k in the sequential algorithm. At that stage we have to merge two linear subtrees consisting of edges $e_{x_1}, \ldots, e_{x_{2^k}}$ and $e_{y_1}, \ldots, e_{y_{2^k}}$. To determine the final position of every element in the resulting block B_{xy} consisting of edges $e_{xy_1}, \ldots, e_{xy_{2^{k+1}}}$, we compare and add up the corresponding positions of an edge $e_{x_i} \in B_x$ that merges in between two edges $e_{y_j}, e_{y_{j+1}} \in B_y$. So if for $e_{x_i} \in B_x$ and $e_{y_j}, e_{y_{j+1}} \in B_y$ holds that $w(e_{y_j}) \leq w(e_{x_i}) \leq w(e_{y_{j+1}})$ the new position-label of e_{x_i} in B_{xy}, then, is $e_{x_i+y_j}$.

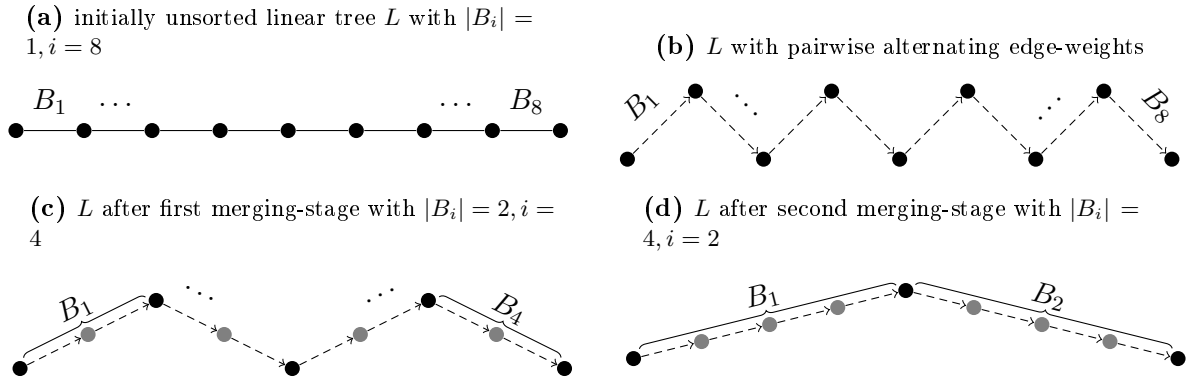

Figure 7: Sorting edges on a linear tree L via merging and pulling out alternating sequences of edge-weights B_i. Note, that the length $|B_i|$ of the sorted sequences doubles with every finalization of a merging-stage.

Algorithm 10: TREE_MERGE_SORT

Input: Linear tree L_{T_1}, permutation π_1 of internal edges of L_{T_1}, permutation π_2 of inner edges of $L_{A_{T_1}}$.

Output: sequence $\mathcal{N}$ of nni-operations that transforms π_1 into π_2

begin

  for $k = 1$ **to** $\log n$ **do**

    foreach $l \in \{1, \ldots, \frac{n}{2 \cdot 2^k}\}$ **parallel do**

      $B_l := merge(B_l, B_{\frac{n}{2^k}-(l-1)})$; `/* Merging two consecutive blocks */`

      $\pi^k := \pi_{B_l} \circ \pi^k$; `/* at the end of the foreach-Phase in the` k`-th iteration,` $\pi^k = e^k_1, \ldots, e^k_{n-3}$ `*/`

    $nni(k) := \big((leaf(e^k_1), e^k_1, e^k_2), \ldots, (leaf(e^k_{n-4}), e^k_{n-4}, e^k_{n-3})\big)$;

    $\mathcal{N} := \mathcal{N} \circ nni(k)$;

end

The number of comparisons used by this variant of the merge sort is in $\mathcal{O}(n \log n)$, although the merging is performed parallelly with an additional number of comparisons. Hence we obtain the following Lemma:

Lemma 4.2. *The number of parallel steps for merge-sorting two linear trees is bounded by* $\Omega(\log n)$.

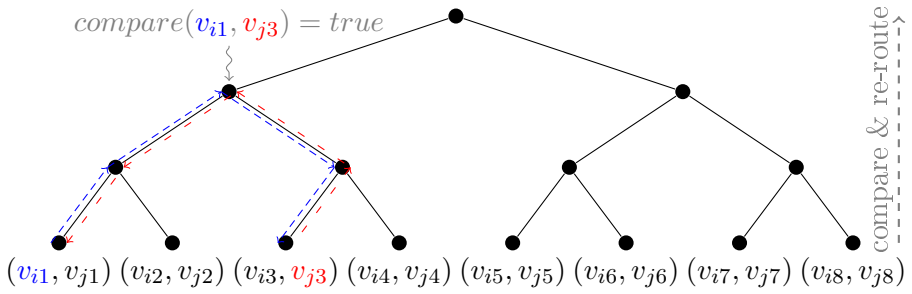

Figure 8: Initial situation on the overlay-graph of T_1' and T_2'. An example re-routing and change of permutation is showed for the matching pair (v_{i1}, v_{j3}) on the left.

Proof. In Algorithm 10 on the previous page the length of the sorted sub-sequences $|B_l|$ doubles with every merging-stage. Therefore at most $\log n$ complete merging-rounds are needed to yield a sorted sequence of length n. □

In the following main step, called Phase 2, the methods described above are used to transform the second phylogeny T_2 into its balanced binary shape T_2' with a common internal topology. The next section deals with the correct sequencing of leaves in both, T_1' and T_2'.

4.4. Sorting Leaf-Permutations on Balanced Binary Trees

Subsequent to Phase 1 and 2, we have two balanced binary trees T_1', T_2' with the same ordering on internal edges covering the same set of n leaves. The permutation of leaves will be given by π_1, π_2, and the position of leaf $v \in \mathcal{L}$ in one of the permutations by $\pi_i(v)$ respectively.

Algorithm: Parallel Bottom-up Leaf Sequencing

Our aim is to transform permutation of leaves imposed by π_1 into π_2. Therefore we create an *overlay-graph* of T_1' and T_2' in which leaves $v_i \in T_1'$ and $v_j \in T_2'$ share their position in the graph iff $\pi_1(v_i) = \pi_2(v_j)$. Then we start a parallel bottom-up *compare* and *re-route* processing at leaf level, at first comparing the labels of the two leaves on a shared node. If the leaf labels match we already have correct correspondence at leaf level and we are done. If the labels do not match, we simultaneously move up the non-matching leaves to parent-level by one nni-operation and repeat the comparison with all newly/recently shared leaves at that node. Now, if (at any parent level) two labels match, we virtually perform the inverse nni-moves of the matching-partner leading to that node to shift the leaves into their correct and final positions. See Figure 8 for an example.

Since the height of the balanced binary tree is bounded by $\lceil \log n \rceil$ the number of nni-moves used by our leaf-sequencing scheme is bounded by $\mathcal{O}(\log n)$, moving every leaf once up and down the tree. Furthermore the following holds:

Lemma 4.3. *The number of comparison-phases is bounded by* $\lceil \log n \rceil$.

Proof. Since the parallel comparison is performed at every level of the tree, the number of complete comparison-phases is bounded by $\lceil \log n \rceil$, e.g. the height of the tree. □

Algorithm 11: PARALLEL_LEAF_SEQUENCING

Input: Overlay-graph $O_i(V^*, E)$ of T_i' with shared nodes at leaf level
$V^* = (V \setminus \mathcal{L}) \cup \{(v_i, v_j) | \pi_1(v_i) = \pi_2(v_j)\}$.

Output: Sequence of nni-moves to match π_1 with π_2 within π_{final}.

begin
  $pos(v) := \pi(v)$; /* initialize position marker */
  for $k = 0, \ldots, (\log n - 1)$ **do**
    foreach $p = 1, \ldots, \frac{n}{k \cdot 2}$ **parallel do**
      parallel_compare($\{v_{ip}\}, \{v_{jp}\}$); /* compare sets on same position */
      if $compare(v_{il}, v_{jl}) = true$ **then**
        $\pi_{final}(v_{il}) := \pi_2(v_{jl})$; /* swap to target position */
        $\pi_{final}(v_{jl}) := \pi_1(v_{il})$;
        $V^* := V^* \setminus \{v_{il}, v_{jl}\}$; /* reduce active set */
      foreach $v_{il} \in \{v_{ip}\}$, $v_{jl} \in \{v_{jp}\}$ **parallel do**
        $pos(v_{il}) := \lfloor \frac{pos(v_{il})}{2} \rfloor$; /* level-up non-matching leaves */
        $pos(v_{jl}) := \lfloor \frac{pos(v_{jl})}{2} \rfloor$;
end

5. Summary

We have developed a new method and design of an efficient parallel algorithm based on Das-Gupta's approximation algorithm for computing the nearest-neighbor-interchange-distance (nni) between weighted phylogenies. The formulations are given in terms of NC-algorithms and the PRAM-model. It is shown that the sequence of nni-operations can be computed efficiently in $O(\log n)$ time on a CRCW-PRAM with a polynomial number of processors and within approximation ratio $4(1 + \log n)$.

References

[Day85] W.H.E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2(1):7–28, 1985.

[DHJ+97] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang. On distances between phylogenetic trees. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 427–436. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1997.

[DHJ+99] B. DasGupta, X. He, T. Jiang, M. Li, and J. Tromp. On the linear-cost subtree-transfer distance between phylogenetic trees. *Algorithmica*, 25(2):176–195, 1999.

[DHJ+00] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang. On computing the nearest neighbor interchange distance. In *Proc. DIMACS Workshop on Discrete Problems with Medical Applications*, volume 55, pages 125–143. Press, 2000.

[GKP94] S. Goddard, S. Kumar, and J.F. Prins. Connected components algorithms for mesh-connected parallel computers. *Parallel Algorithms: 3rd DIMACS Implementation Challenge*, 30:43–58, 1994.

[Hir76] DS Hirschberg. Parallel Algorithms for the Transitive Closure and the Connected Components Problems Proc. 8thAnn. In *ACM Symp. Th. Comp*, pages 55–57, 1976.

[LTZ96] M. Li, J. Tromp, and L. Zhang. Some notes on the nearest neighbour interchange distance. *Lecture Notes in Computer Science*, pages 343–351, 1996.

[RF79] D. Robinson and L. Foulds. Comparison of weighted labeled trees. In *Combinatorial Mathematics VI: Proceedings of the Sixth Australian Conference on Combinatorial Mathematics, Armidale, Australia, August 1978*, page 119. Springer, 1979.

[Rob71] DF Robinson. Comparison of labeled trees with valency three. *Journal of Combinatorial Theory*, 11(2):105–119, 1971.