Politechnika Gdańska

WYDZIAŁ ELEKTRONIKI
TELEKOMUNIKACJI I INFORMATYKI

Katedra: **Algorytmów i Modelowania Systemów**

Imię i nazwisko dyplomanta: **Anna Pawelczyk**

Nr albumu: **106391**

Forma i poziom studiów: **dzienne, stacjonarne, magisterskie**

Kierunek studiów: **Informatyka**

Praca dyplomowa

Temat pracy: Metody definiowania dystansów dla drzew filogenetycznych

Opiekun pracy: dr hab. inż. Krzysztof Giaro, prof. nadzw. PG

Zakres pracy:
Cele pracy to:

- wytworzenie efektywnego oprogramowania do porównywania drzew filogenetycznych, implementującego klasyczne metryki i metryki opracowane na wydziale Informatyki Politechniki Gdańskiej, opierające się na znajdywaniu doskonałego skojarzenia w grafie,
- analiza zaimplementowanych rozwiązań z wypracowaniem uniwersalnego modelu konstrukcji metryk drzew filogenetycznych.

Zadaniem niniejszego dokumentu jest prezentacja zaimplementowanej biblioteki do porównywania drzew filogenetycznych *PhylotreeDist*, jej funkcjonalności, zastosowania i wymagań oraz kompleksowe przedstawienie zagadnień związanych z biblioteką, do których należą:

- Filogeneza
- Algorytmy porównywania drzew filogenetycznych
- Analiza istniejących bibliotek filogenetycznych

Gdańsk, 2011 rok

OŚWIADCZENIE

Oświadczam, że niniejszą pracę dyplomową wykonałem samodzielnie. Wszystkie informacje umieszczone w pracy uzyskane ze źródeł pisanych oraz informacje ustne pochodzące od innych osób zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami.

.....

podpis dyplomanta

Beksiński, 1988, [101]

Even if I knew that tomorrow the world would go to pieces,
I would still plant my apple tree.

Martin Luther

Contents

Chapter 1

Abstract

The amount of data that is used to estimate relationships among species requires computer computations which base on diverse algorithms of sequence alignment and phylogenetic trees construction. This diversification in conjunction with the constantly augmenting knowledge about genomic sequences results in generating many hypothetical phylogenetic trees or parts of trees. The field of comparing phylogenetic trees is the domain of this work.

The aim of this paper is to give an outline of the phylogenetic trees comparison library *PhylotreeDist*, drawing a special attention to one of the algorithms the library implements - the *Minimum Weight Perfect Matching* method of trees comparison which is elaborated at Gdansk University of Technology, Poland. Along with the description of *PhylotreeDist*'s usage, functionality and requirements, we provide a comprehensive theoretical background to the library, documenting the researches on the three areas that directly concern the library: phylogenetics, algorithms for phylogenetic trees comparison and existing bioinformatic solutions.

Key words

bioinformatics, phylogenetics, phylogenetic trees distance, phylogenetic trees metric, bioinformatic library, C++, Robinson-Foulds, quartet distance, triplets distance, nodal distance, minimum weight perfect matching distance, PhylotreeDist, Bio++.

Chapter 2

Streszczenie w języku polskim

Niniejszy dokument jest teoretyczną częścią Pracy Magisterskiej *Metody definiowania dystansów dla drzew filogenetycznych* (ang. *The Methods of Phylogenetic Trees Distance Definition*) Anny Pawelczyk, studentki wydziału Elektroniki, Telekomunikacji i Informatyki (ETI) na Politechnice Gdańskiej. Promotorem pracy jest dr inż. Krzysztof Giaro, profesor nadzwyczajny Politechniki Gdańskiej katedry Algorytmów i Modelowania Systemów.

Praca ta składa się z tego dokumentu i bioinformatycznej biblioteki *PhylotreeDist* służącej do wyznaczania dystansów dla drzew filogenetycznych.

Wprowadzenie

Współczesny model odtwarzania historii organizmów żywych oparty jest na teorii ewolucji gatunków. Wszystkie organizmy żywe wywodzą się od wspólnego przodka, a różnice w ich charakterystyce spowodowane są mutacjami materiału genetycznego. Rodowód gatunków wyższych przedstawiany jest wg pomysłu Karola Darwina jako drzewo filogenetyczne.

Zakres danych jakie są potrzebne do oszacowania pokrewieństwa gatunków wymaga zastosowania obliczeń komputerowych, które bazują na różnorodnych algorytmach porównywania sekwencji genetycznych i konstrukcji drzew filogenetycznych. Ta dywersyfikacja w algorytmach oraz ciągle rozszerzająca się wiedza na temat materiału genetycznego organizmów żywych prowadzi do otrzymywania wielu hipotetycznych drzew filogenetycznych lub fragmentów tych drzew. Na przeciw temu zagadnieniu wychodzą **metryki do porównywania drzew filogenetycznych**.

Cel

Celem pracy *Metody definiowania dystansów dla drzew filogenetycznych* jest wytworzenie efektywnego oprogramowania do porównywania drzew filogenetycznych implementującego klasyczne metryki i metryki opracowane na wydziale Informatyki Politechniki Gdańskiej a także analiza zaimplementowanych rozwiązań z wypracowaniem uniwersalnego modelu konstrukcji metryk drzew filogenetycznych.

Zadaniem niniejszego dokumentu jest prezentacja biblioteki do porównywania drzew filogenetycznych *PhylotreeDist*, jej funkcjonalności, zastosowania i wymagań oraz kompleksowe przedstawienie zagadnień związanych z biblioteką, do których należą:

- **Filogeneza**

  Zarys wiedzy na temat filogenezy, w tym jej historia, zastosowanie i podstawowe algorytmy konstrukcji drzew filogenetycznych.

- **Algorytmy porównywania drzew filogenetycznych**

  Kompleksowa analiza teoretyczna wszystkich metryk zaimplementowanych w *PhylotreeDist* z podziałem na popularne metryki stosowane w bibliotekach bioinformatycznych oraz nowe, opracowane na wydziale Informatyki Politechniki Gdańskiej podejście do obliczania dystansu między drzewami z użyciem algorytmu znalezienia minimalnej wagi doskonałego skojarzenia grafu. Analiza metryk zawiera definicję, przybliżenie Czytelnikowi definicji za pomocą intuicyjnego algorytmu z analizą jego złożoności czasowej, dowód poprawności definicji jako metryki, analizę współcześnie najlepszych (na rok 2011) algorytmów dla metryki, opis niektórych szczegółów implementacyjnych algorytmów oraz przykłady.

  Analiza metryk prowadzi do jednego z ważniejszych produktów tej pracy: wypracowania ogólnego modelu konstrukcji dystansów drzew filogenetycznych - meta-podejścia do definiowania dystansów między drzewami.

- **Analiza istniejących bibliotek filogenetycznych**

  Zestawienie informacji na temat bibliotek filogenetycznych wg dwóch kryteriów charakteryzujących *PhylotreeDist*: biblioteki bioinformatyczne w języku C++ oraz biblioteki, które implementują algorytmy porównywania drzew filogenetycznych. Analiza bibliotek prowadzi również do wyboru i bardziej szczegółowego opisu biblioteki *Bio++* [40], której

struktury przechowujące drzewa filogenetyczne są rozwiązaniem użytym w *PhylotreeDist*.

Zawartość

Niniejszy dokument podzielony jest na wstęp i 3 części (nazwy działów przedstawione są w języku polskim i angielskim z użyciem skrótu *ang.*). Dokument zakończony jest rozdziałem **13 Podsumowanie i wyniki** (ang.: **Summary and Results**), w którym zebrane są wyniki pracy i wnioski jakie powstały podczas jej tworzenia.

We wstępie pracy przedstawiony jest zarys dziedziny pracy w języku angielskim (**rozdział 1**, ang.: **Abstract**), streszczenie w języku polskim (**rozdział 2**), wprowadzenie (**rozdział 3**, ang.: **Introduction**) oraz wprowadzone są definicje pojęć wykorzystywanych w pracy, będących elementami dziedzin: teoria grafów, matematyka, filogeneza (**rozdział 4**, ang.: **Basic Definitions**).

Część I Podstawy filogenezy i jej zastosowanie
(ang.: **Part I Basics and Application of Phylogenesis**)

W tej części przybliżone jest pojęcie *filogenezy* i omówione są jej zastosowania w różnych dziedzinach. **Rozdział 5 Wprowadzenie do filogenezy** (ang: **Introduction to Phylogenesis**) zwięźle omawia filogenezę, przedstawiając jej historię i podstawy. W **rozdziale 6 Jak drzewo życia wpływa na nasze życie** (ang: **How the Tree of Life Impacts Our Life**) przedstawione są różne dziedziny, również te niekoniecznie oczywiste, w które korzystają z wiedzy dostarczanej przez filogenezę.

Część II Algorytmy wyznaczania dystansów między drzewami filogenetycznymi
(ang.: **Phylogenetic Trees Distances Algorithms**)

Część ta zawiera teorię porównywania drzew filogenetycznych. Zasadniczym jej rezultatem jest opracowanie modelu popularnych metryk i ich klasyfikacja wg tego modelu. Ponadto w **rozdziale 7 Metryki klasyczne** (ang.: **Classical Metrics**) zebrane są wiadomości o popularnych metrykach dla drzew filogenetycznych, w tym ich definicje, intuicyjne algorytmy z analizą złożoności czasowej, analiza współcześnie najlepszych algorytmów z opisem

niektórych szczegółów implementacyjnych oraz przykłady. **Rozdział 8 Dystansy minimalnej wagi doskonałego skojarzenia grafu** (ang.: **Minimum Weight Perfect Matching Distance**) zawiera opis rodziny metryk bazujących na algorytmie znalezienia minimalnej wagi doskonałego skojarzenia grafu opisującego drzewa. W rozdziale zawarte są również definicje trzech instancji tej rodziny metryk z ich analizą, podobnie jak w poprzednim rozdziale. Dla wszystkich metryk opisanych w części II przeprowadzone są dowody ich zgodności z definicją metryki.

Część III Biblioteka PhylotreeDist
(ang.: **Part III The PhylotreeDist Library**)

Ta część poświęcona jest bibliotece *PhylotreeDist* oraz innym bibliotekom z nią związanym. Biblioteka przedstawiona jest w kontekstach:

- funkcjonalność i zastosowanie (**rozdziały 11 PhylotreeDist**, **12 PhylotreeDist manual**).
- architektura i zależności (**rozdział 11 PhylotreeDist**).
- analiza Bio++ - biblioteki dostarczającej implementację drzewa filogenetycznego dla PhylotreDist (**rozdział 10 PhylotreeDist jako plugin do biblioteki filogenetycznej Bio++**, ang.: **PhylotreeDist as a Plugin to Bio++ Phylogenetic Library**).
- omówienie różnych bibliotek filogenetycznych i szerzej - bioinformatycznych w dwóch różnych kontekstach: bibliotek implementujących dystanse drzew filogenetycznych oraz popularnych bioinformatycznych bibliotek w języku C++, w którym to języku jest również zaimplementowana PhylotreeDist (**rozdział 9 Istniejące implementacje**, ang.: **Existing Solutions**).

Wyniki i wnioski

Niniejsza praca wprowadza cztery podstawowe produkty:

1. implementacja dystansów minimalnej wagi doskonałego skojarzenia grafu dla drzew filogenetycznych i ich szczegółowa analiza teoretyczna
2. implementacja i analiza teoretyczna klasycznych algorytmów porównywania drzew filogenetycznych z naciskiem na wykorzystanie algorytmów o najlepszych złozonościach czasowych i optymalizację kodu
3. wypracowanie ogólnego modelu konstrukcji dystansów drzew filogenetycznych
4. sugestie udoskonalenia biblioteki Bio++

Ponadto, dokument zawiera analizę:

- metryk do porównywania drzew filogenetycznych, dowody ich poprawności w sensie definicji metryki oraz analizę algorytmów i ich złożoności
- zastosowania filogenezy
- bibliotek bioinformarycznych w języku C++ oraz bibliotek implementujących dystansy drzew filogenetycznych
- biblioteki Bio++, w szczególności jej pakietu filogenetycznego

Praca ta jest bazą dla przyszłych działań:

1. badań nad dystansami drzew filogenetycznych, zwłaszcza bazujących na metryce doskonałego skojarzenia grafu rozwijanej na wydziale Informatyki Politechniki Gdańskiej
2. dodania biblioteki PhylotreeDist do bazy pakietów bioinformatycznych laboratorium Felsenstein/Kuhner [86]
3. dołączenia algorytmów zaimplementowanych w PhylotreeDist do otwartej biblioteki Bio++
4. sugestii dla Bio++ związanych z jej kodem i wydajnością

PhylotreeDist

Biblioteka *PhylotreeDist* jest podstawowym użytkowym wynikiem tej pracy magisterskiej. Jest to bioinformatyczne narzędziem w języku C++, którego dwoma nadrzędnymi celami są:

- implementacja metryk dla drzew filogenetycznych opracowanych na wydziale ETI Politechniki Gdańskiej, których działanie opiera się na znalezieniu najtańszego doskonałego skojarzenia grafu opisującego dwa drzewa filogenetyczne,
- zgromadzenie najpopularniejszych metryk dla drzew filogenetycznych (nazywanych metrykami *klasycznymi*) z naciskiem na zaimplementowanie ich najefektywniejszych algorytmów.

Oprogramowanie to ma być darmowe, mieć otwarty kod. W celu popularyzacji, jest ono zgodne z interfejsem bioinformatycznej biblioteki Open Source *Bio++*, której planowane jest zaproponowanie kodu *PhylotreeDist*.

W tablicy 11.1 przedstawiona jest lista zaimplementowanych algorytmów z ich krótką charakterystyką. Opis każdego algorytmu zawiera jego złożoność obliczeniową i pamięciową, wymogi dla drzew, na których operuje oraz referencje do literatury na bazie której został zaimplementowany i opisującej go sekcji niniejszego dokumentu.

Angielskie nazwy *quartets, triplets, nodal* można tłumaczyć jako dystanse: kwartetowy, trójkowy, węzłowy. *MWPM* - z ang. Minimum Weight Perfect Matching - algorytm najtańszego doskonałego skojarzenia grafu.

metryka	typ drzew*	złożoność		literatura
		czasowa	pamięciowa	
Robinson-Foulds	dowolne	$O(n)$	$O(n)$	sekcja 7.2 , [19]
ważony Robinson-Foulds	nieukorzenione z wagami na gałęziach	$O(n)$	$O(n)$	sekcja 7.2 , [19]
Quartets	nieukorzenione	patrz **	$O(n^2)$	sekcja 7.3 , [15]
Triplets	ukorzenione	$O(n^2)$	$O(n^2)$	sekcja 7.4 , [18]
Nodal (manhattan)	nieukorzenione	$O(n^2)$	$O(n)$	sekcja 7.5 , [4]
ważony Nodal (manhattan)	nieukorzenione z wagami na gałęziach	$O(n^2)$	$O(n)$	sekcja 7.5 , [4]
Nodal (pitagorejski)	nieukorzenione	$O(n^2)$	$O(n)$	sekcja 7.5 , [4]
ważony Nodal (pitagorejski)	nieukorzenione z wagami na gałęziach	$O(n^2)$	$O(n)$	sekcja 7.5 , [4]
MWPM na splitach	nieukorzenione	$O(n^3)$	$O(n^2)$	sekcja 8.2 , [7]
MWPM na klustrach	ukorzenione	$O(n^3)$	$O(n^2)$	sekcja 8.3 , [7]
MWPM na parach	ukorzenione binarne	$O(n^3)$	$O(n^2)$	sekcja 8.3 , [7]

Table 2.1: Algorytmy implementowane przez bibliotekę *PhylotreeDist*.
*Porównywane drzewa muszą mieć ten sam zbiór liści.
** $O(|L| + |V_1 \setminus L|\ |V_2 \setminus L|\ min\{\Delta(T_1 \setminus L), \Delta(T_2 \setminus L)\})$ dla drzew $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ o zbiorze liści L.

Interfejsem biblioteki jest klasa *dist :: PhylotreeDist*. Dla każdej z metryk zaimplementowana jest w niej statyczna metoda obliczająca odległości między dwoma drzewami przekazanymi jako parametry wywołania. Tablica 11.2 prezentuje nazwy metod wszystkich zaimplementowanych w klasie *dist::PhylotreeDist* metryk.

metryka	nazwa metody
Robinson-Foulds	robinsonFoulds
ważony Robinson-Foulds	robinsonFouldsW
Quartet	quartetDistance
Triplets	tripletsDistance
Nodal (manhattan)	nodalDistance
ważony Nodal (manhattan)	nodalDistanceW
Nodal (pitagorejski)	nodalDistance_pythagorean
ważony Nodal (pitagorejski)	nodalDistanceW_pythagorean
MWPM na splitach	perfectMatching_splits
MWPM na klustrach	perfectMatching_clusters
MWPM na parach	perfectMatching_pairs

Table 2.2: Nazwy metod klasy *dist::PhylotreeDist*.

Drzewa, które przekazywane są jako parametry to obiekty typu *const bpp::TreeTemplate<Node>* zaimplementowane w bibliotece *Bio++*. Ponadto Użytkownik może zdecydować za pomocą flag przekazywanych jako parametry funkcji, czy metoda ma wykonać testy zgodności drzew wejściowych z wymaganiami dla metryki (funkcjonalność jest opcjonalna, jako że bywa kosztowna czasowo). Odległość między drzewami zwracana jest jako wartość funkcji.

Poniżej przedstawione są dwa możliwe szablony sygnatury, które spełniają metody klasy *dist::PhylotreeDist*. Wybór szablonu zależy od tego, czy wartość obliczana przez metrykę czyli zwracana przez funkcję jest liczbą całkowitą czy zmiennoprzecinkową.

```
static int dist::PhylotreeDist::_nazwa_metryki (
        const bpp::TreeTemplate< bpp::Node > & trIn1,
        const bpp::TreeTemplate< bpp::Node > & trIn2,
        bool setNodesId = true,
        bool checkNames = false
) throw (bpp::Exception)
```

```
static double dist::PhylotreeDist::_nazwa_metryki (
        const bpp::TreeTemplate< bpp::Node > & trIn1,
        const bpp::TreeTemplate< bpp::Node > & trIn2,
        bool setNodesId = true,
        bool checkNames = false
) throw (bpp::Exception)
```

gdzie:

[in] **tr1** Pierwsze porównywane drzewo.

[in] **tr2** Drugie porównywane drzewo.

[in] **setLeavesId** Argument opcjonalny. Należy ustawić wartość na *true*, jeśli drzewa wejściowe nie zostały przed podaniem jako argumenty przekształcone tak, by miały te same numery id dla liści o tych samych nazwach lub jeśli numery id liści nie są ciągiem liczb $0..n-1$. Wartość domyślna: *true*.

[in] **checkNames** Argument opcjonalny. Należy ustawić wartość na *true*, jeśli użytkownik chce sprawdzić, czy drzewa wejściowe spełniają wymogi metryki. Wartość domyślna: *false*.

return zwraca dystans pomiędzy drzewami

throw (bpp::Exception) wyjątek wyrzucony jeśli porównywane drzewa nie spełniają wymogów metryki

Słowa kluczowe

bioinformatyka, filogeneza, metryka porównująca drzewa filogenetyczne, dystanse drzew filogenetycznych, biblioteka bioinformatyczna, C++, Robinson-Foulds, dystans kwartetowy, dystans tripletowy, dystans węzłowy, dystans minimalnej wagi doskonałego skojarzenia grafu, PhylotreeDist, Bio++.

Kiedy byłem mały
Pytałem co to życie, pytałem co to jest życie mamo
- Widzisz życie to...

zjawisko biologiczne złożone i wielowymiarowe, którego nie można opisać za pomocą jednej prostej definicji. Znane dotychczas wyłącznie z Ziemi i w tym kontekście definiowane w odniesieniu do 2 podstawowych znaczeń: 1) na określenie stanu materii (nazywanej organizmem) trwającego od pojawienia się (narodzin) organizmu do zakończenia jego bytu osobniczego, w większości kończącego się śmiercią (biol.); 2) na określenie dynamicznego procesu, który pojawił się na Ziemi ok. 3,8 mld lat temu, obejmującego pochodzące od jednej formy wyjściowej wszystkie istniejące w przeszłości i żyjące obecnie organizmy wraz z wszelkimi wzajemnymi relacjami i zależnościami oraz ich wpływem na środowisko.

- mamo... nie rozumiem...
- widzisz... *widzisz życie to ja i Ty*
Ten ptak to drzewo i kwiat
Odpowiadała mi

W tekście wykorzystano
frgm. utw. *Naiwne pytania* zespołu Dżem, słowa Ryszard Riedel [88]
oraz definicję hasła *życie* z Encyklopedii PWN 2010, [46]

Chapter 3

Introduction

3.1 Background

This paper is the theoretical part of a Master Thesis by Anna Pawelczyk, student at Gdansk University of Technology, Poland, ETI Faculty, advised by PhD DSc Krzysztof Giaro, Associate Professor at Gdansk University of Technology, ETI Faculty, Dept. of Algorithms and System Modeling.

The thesis comprises of this document and a phylogenetic trees distances library *PhylotreeDist*.

3.2 Goal

The aim of this paper is to give an outline of the phylogenetic trees comparison library *PhylotreeDist*, drawing a special attention to one of the algorithms the library implements - the *Minimum Weight Perfect Matching* method of trees comparison which is elaborated at Gdansk University of Technology, Poland.

Along with the description of PhylotreeDist usage, functionality and requirements, we provide a comprehensive theoretical background to the library, documenting the researches on the three areas that directly concern the library:

1. Phylogenetics:
   a sketch of the phylogeny topic, including its history, application and basis of constructing the phylogenetic trees.

2. Algorithms for phylogenetic trees comparison:
   the methodical analysis of all the algorithms implemented in the PhylotreeDist library. For each metric, we provide its definition and bring it closer by intuitive algorithm; afterwards we present and analyse the best known algorithm for the method, giving some implementation shortcuts and picturing the theory with example. We also prove the correctness of metrics definitions.

3. Existing bioinformatic solutions:
   the review of Bioinformatic libraries that, just as the PhylotreeDist does, implement trees distances or that also are written in C++. The examination of the libraries results in a choice of Bio++ library as the provider of tree-structure objects.

3.3 Content

This document consists of introducing chapters and the three parts that present the results of the researches: Basics and Application of Phylogenesis, Phylogenetic Trees Distances Algorithms, and The PhylotreeDist Library. The paper is summarized with **chapter 13 Summary and Results**, which gathers the findings and ideas and that arose during the researches and PhylotreeDist implementation and future goals.

Introducing chapters
The introducing chapters consist of the **abstract** (**chapter 1**), summary of the work in the authors' native Polish language (**chapter 2 Streszczenie w języku polskim**), **introduction** (**chapter 3**) and the **chapter 4 Basic Definitions** where the terms used in this paper are introduced with respect to the following categories: graph theory, mathematical appliance, phylogenesis.

Part I Basics and Application of Phylogenesis
In this part we introduce the term *Phylogenesis*, drawing special attention to exposition of beneficial application of this field of science. In the **chapter 5**

Introduction to Phylogenesis we briefly introduce the world of Phylogenesis by presenting the history of this discipline and briefly characterizing its basis. In the **chapter 6 How the Tree of Life Impacts Our Life** we present the not evident fields that benefit from Phylogenetic knowledge.

Part II Phylogenetic Trees Distances Algorithms
This is the theoretical, algorithms-oriented part, strictly concerned on phylogenetic trees distances. Here we present the idea of what the phylogenetic trees metric is. We also introduce the meta-attitude to trees comparison metrics structure, presenting general procedures common for all the most popular trees comparison metrics as well as for the minimum weight perfect matching metrics.

Moreover, in the **chapter 7 Classical Metrics** we collect information on well-known phylogenetic metrics. This includes general algorithms description as well as detailed presentation of their best known (for the year 2011) time complexity algorithms. **Chapter 8 Minimum Weight Perfect Matching Distance** provides general assumptions and proceedings for the family of metrics that are called as is the chapter title. Next we present the family's special cases, giving their detailed definitions and algorithms. We also prove the metric properties for all the methods.

Part III The PhylotreeDist Library
The focus of this part is the *PhylotreeDist* - the library implemented as a part of this thesis. The library is presented in context of:

- library's functionalities and usage (**chapter 11 PhylotreeDist**, **chapter 12 PhylotreeDist manual**).
- architecture and dependencies of the library (**chapter 11**).
- closer look on the Bio++ - the library that PhylotreDist depends on (**chapter 10 PhylotreeDist as a Plugin to Bio++ Phylogenetic Library**).
- an outline of other phylogenetic libraries in two contexts: what trees distance methods the bioinformatic libraries provide and popular C++ source code language libraries (**chapter 9 Existing Solutions**).

Chapter 4

Basic Definitions

4.1 Graph theory

A **graph** is a pair $G = (V, E)$ in which V is a finite set of **vertices** and E is a finite set of **edges** as unordered pairs $\{v, u\} \in E$, where $u, v \in V$. If $\{u, v\}$ is an edge of G, then u and v are **adjacent**. A **degree** $deg(v)$ of a vertex v is the number of edges that end at that vertex. The maximal degree over all the vertices in a graph G is denoted as $\Delta(G)$. We say that a graph $G' = (V', E')$ is a **subgraph** of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. [25] [17] [11]

A **walk** in a graph is a sequence $W = v_0e_1v_1e_2...v_{k-1}e_kv_k$ of alternatively vertices and edges such that for $1 \leq i \leq k$ the ends of e_i are v_{i-1}, v_i. A **path** is a walk where for vertices v_i, v_j, $i \neq j$ there is satisfied a condition $v_i \neq v_j$. A graph contains a **cycle** if it contains a finite non-null walk $C = v_0e_1v_1e_2...v_{k-1}e_kv_0$ such that the ends of e_k are v_{k-1}, v_0 and $v_i \neq v_j$ for $0 \leq i < j \leq k - 1$. A graph with no cycles is called **acyclic**. A graph is **connected** if all its vertices are connected e.g. for each $u, v \in V$ there is a path between u, v. [8] [30]

A **complete graph** is an undirected graph in which every pair of vertices is adjacent. A **bipartite graph** $G(V, E)$ is a graph in which V can be partitioned into two sets V_1 and V_2 such that $\{u, v\} \in E$ implies either $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$. That is, all edges go between two sets V_1 and V_2. [17] A **complete bipartite graph** is a special kind of bipartite graph where every vertex from the V_1 set is connected to every vertex from the V_2 set.

Given an undirected graph $G = (V, E)$, a **matching** is a subset of edges

$M \subseteq E$ such that for all vertices $v \in V$, at most one edge of M is incident on v. A **perfect matching** is a matching in which every vertex is matched. [17] The **assignment problem** is to find a **Minimum Weight Perfect Matching in Bipartite Graphs** is when given a bipartite graph $G = (V, E)$ with bipartitions (V_1, V_2) and weight function $w : E \to \mathbb{R}$, to find a perfect matching M minimizing $w(M) = \Sigma_{e \in M} w(e)$. [84]

A **tree** is an acyclic connected graph. Its vertices are also called **nodes**. The vertices with degree one are called **leaves**. For the purpose of this paper, a trees has every leaf labelled uniquely, the set of leaves is called L, the tree built on the L is T_L. A vertex that is not a leaf is called an **internal vertex**. An edge that is not adjacent to a leaf is called an **internal edge**. The number of all tree's edges is $|E| = |V| - 1$. [12]

A **rooted tree** is a tree with one internal vertex distinguished and called the **root**. Each edge is implicitly directed away from the root. Designating a root imposes a hierarchy on the vertices of a rooted tree according to their distance from that root. The **level** of a vertex v is its distance from the root, i.e. the number of edges of the unique path from the root to v. The **height** of a rooted tree is the length of the longest path from the root. If a vertex v immediately precedes a vertex w on the path from the *root* to w, then v is a **parent** of w and w is a **child** (also called a **son**) of v. Vertices having the same parents are called **siblings**. If a vertex u is on the unique path from the root to a vertex y, then u is called an **ancestor** of y and y is a **descendant** of u. [12]

An **unrooted tree** is a tree with no vertices of degree two and no root node. [12]

A **binary tree**, also called **bifurcating** or **fully resolved** is defined as follows: for unrooted trees - the tree which every internal vertex has degree three; for rooted trees: the tree which every internal vertex has degree three except a root, which has degree two; every binary rooted tree of height h has at most $2^{h+1} - 1$ vertices. In binary unrooted trees always $|V| = 2|L| - 2$, in binary rooted trees always $|V| = 2|L| - 1$. For sample trees see figure 4.1. [12]

A **multifurcating tree** is a tree that is not binary; which internal nodes have an arbitrary, possibly different degree. [12]

The graph search algorithms traverse the tree for finding a vertex with specified properties. The **depth-first search - DFS** algorithm as its name implies, searches "deeper" in the graph whenever possible. In depth-first search,

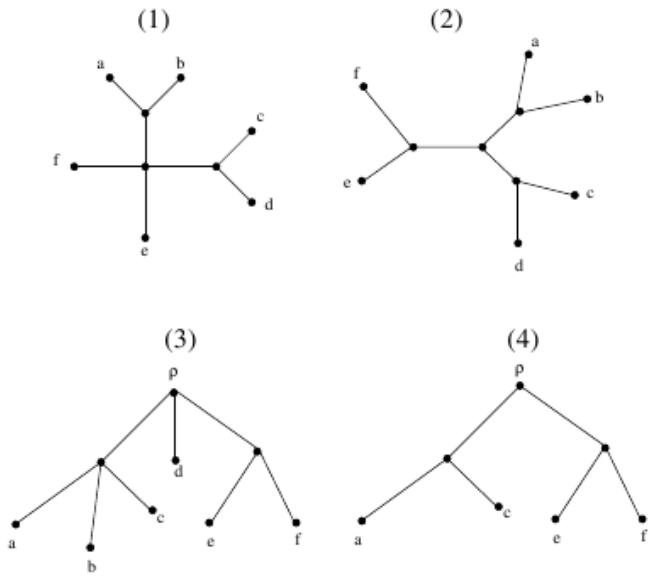


Figure 4.1: Four examples of trees: (1) and (2) are unrooted, (3) and (4) are rooted. (2) and (4) are binary. [12].

edges are explored out of the most recently discovered vertex v that still has unexplored edges leaving it. When all of v's edges have been explored, the search "backtracks" to explore edges leaving the vertex from which v was discovered. This process continues until we have discovered all the vertices that are reachable from the original source vertex. If any undiscovered vertices remain, then one of them is selected as a new source and the search is repeated from that source. This entire process is repeated until a searched vertex is found or until all vertices are discovered.

Given an unrooted tree T on the set of leaves $|L|$ and denoting e to be an edge of $T : e \in E(T)$, if we remove e then we divide up T into two components. Let A be the set of leaves in one component and B be the leaves in the other component. Then $A|B$ is a partition of L into two blocks, called a **split** or **bipartition** of L. If $|A| = 1$ or $|B| = 1$ then $A|B$ is **trivial**, otherwise it is **non-trivial**. In this paper we denote a set of all non-trivial splits of T as $\beta(T)$. The number of non-trivial splits is equal to the number of internal edges. A tree can be reconstructed in linear time from its set of splits. [12]

Given a rooted tree T on the set of leaves $|L|$, if we choose a vertex $v \in V(T)$ then the set of leaves in T that are descendants of v (including v if it is a leaf) is called a **cluster**. A cluster A is **trivial** if $|A| = 1$. In this paper we denote

a set of all non-trivial clusters of T as $\sigma(T)$. The number of non-trivial splits is equal to the number of internal vertices. A rooted tree is uniquely defined by its clusters, and can be reconstructed from these in linear time. [12]

4.2 Mathematical appliance

Let X be an arbitrary set. A function $d : X \times X \to \mathbb{R}$ is a **metric** on X if the following conditions are satisfied for all $x, y, z \in X$:

1. Positiveness: $d(x, y) > 0$ if $x \neq y$, and $d(x, x) = 0$.
2. Symmetry: $d(x, y) = d(y, x)$.
3. Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$.

A **metric space** is a pair (X, d) where d is a metric on X, therefore $d : X \times X \to \mathbb{R}$. Elements of X are called points of the metric space; $d(x, y)$ is referred to as the **distance** between points x and y. Informally, a metric space is a set with a metric on it. [13] A **distance measure** does not have to satisfy the triangle equality condition.

Any set $\mathbb{V}$ on which the operations of vector addition and multiplication by a scalar are defined is said to be a **vector space**. A mapping

$$f \to ||f||, f \in \mathbb{V}, ||f|| \in \mathbb{R}$$

of a vector space $\mathbb{V}$ onto the set of real numbers is called a **norm** if it satisfies the following conditions([27], page 20.):

1. $||f|| > 0$ for $f \neq 0$
2. $||0|| = 0$
3. $||af|| = |a|||f||$ for all $a \in \mathbb{R}$
4. $||f + g|| \leq ||f|| + ||g||$ (triangle inequality)

A **p-norm** where $p \geq 1$ for $x \in \mathbb{V}$ is defined as follows [85]:

$$||x||_p = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$$

4.3 Phylogenesis

Phylogeny is a classification scheme that indicates the evolutionary relationships between **taxa**, where a **taxon** is ant named group of organisms. [11] [91]

Phylogenetic tree, also called a **tree of life** or an **evolutionary tree** represents the phylogeny of organisms, i.e., the history of organismal lineages as they change through time. It implies that different species arise from previous forms via descent, and that all organisms, from the smallest microbe to the largest plants and vertebrates, are connected by the passage of genes along the branches of the phylogenetic tree that links all of Life. [91].
It is represented as a tree topology rooted graph with taxa being represented by leaves and internal nodes representing organisms or DNA sequences that are ancestral to studied taxa. It is often assumed that the tree is bifurcating. Sometimes, the phylogenetic tree is represented as unrooted tree that illustrates relationships between the organisms or DNA sequences being studied, without providing information about the past evolutionary events that have occurred. [11]

Consensus trees are a way to summarise the agreement between two or more trees. The **strict** consensus tree contains only those clusters found in all the trees [54]

Part I

Basics and Application of Phylogenesis

Beksiński, 1976, [99]

Nobody understands the world they're in,
but some people are better off at it than others.

Richard P. Feynman, *The Meaning of it All*

Chapter 5

Introduction to Phylogenesis

5.1 Tree of Life - from the pen's draft to the digital computations

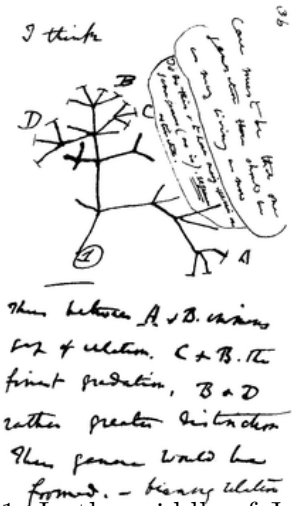


Figure 5.1: In the middle of July 1837 Darwin began to write a notebook "Transmutation of Species". On page 36 over his first evolutionary tree he wrote "I think". [94]

The concept of the Tree of Life – a graph that represents the historical evolutionary relationship between different species or organisms - is dated to be born in July 1837 when, according to tradition, Charles Darwin wrote in his "Transmutation of Species" notebook the first graph of relations among species.

Since that, over the years, scientists have been trying to solve what is the history of all the organisms. Initially, the studies were conducted basing on what was the most reasonable for that times - species morphological features. After the fruitful growth in numerical analysis and computer technology together with the discovery and growth in genetics – it is the organism DNA sequence

that provides basis for estimating the evolutionary relationships between organisms.

Nowadays, thanks to the work of biologist that sequence more and more genomes and computer scientists' achievements in the algorithmic and optimization fields, together with the great advantages of modern computer technology, we can focus on invisible for human's eye changes in the specie's nucleic sequences, mutations, and create the trees that present the evolution process. Afterwards, we compare them, merge and try to approach the history of life.

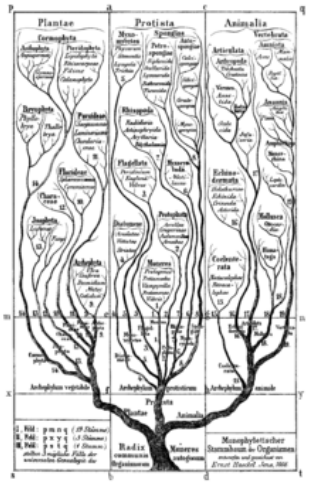


Figure 5.2: One of the first phylogenetic tree. An effect of Ernest Hackle's researches published in his german publication "Generelle Morphologie der Organismen" in 1866. [95]

5.2 Phylogenetic tree's structure

The phylogenetic tree is a regarded as the most convenient way of visualizing evolutionary relationships between a set of taxa. It is a rooted graph of the tree topology which leaves correspond to the final steps of evolution (generally - species) and which internal nodes correspond to speciation events.

A phylogenetic tree has following characteristics (consult the figures 5.3 and 5.4):

- The basic structure is a set of **nodes** (**internal** and **leaves**) and **edges**.

Nodes, also called vertices, represent taxonomic units, such as an organism, a specie, a population, a common ancestor, or even an entire genus or other higher taxonomic group. Nodes can be one of two types:

- a leaf – the final step of evolution.
- an internal node – a speciation event: the point in time where evolution has diverged in different directions. It is interpreted as an ancestor of some taxa: it is explicitly shown of which ones if a tree is rooted.

Edges, also called branches, connect nodes uniquely and represent genetic relationships. The specific pattern of branching determines the tree's topology.

- Rooting
  Rooting the phylogenetic trees is performed to present the **sequence in time** of genomic changes and therefore to determine which taxa are ancestors of which other taxa. The root represents the furthest common ancestor for every taxa. However, as the process of building the tree is very complicated itself, acquiring the undirected tree topology is a valuable step towards achieving the tree of life. Such a tree is induced by ignoring the location of the root and the length of edges.

- Weighted branches
  To mark the proportions of some biological properties such as the number of amino acid changes between nodes on a protein phylogeny or the time passage, phylogenetic trees can have their branches weighted.

- Bifurcation (also: fully resolved trees)
  During the course of life, the spices acquire changes to their DNA e.i. via coping error during DNA replication or because of DNA damage through environmental agents including sunlight, cigarette smoke, and radiation. Some of these changes occur in cells of the body, such as in skin cells as a result of sun exposure, but are not passed on to children. But other errors can occur in the DNA of cells that produce the eggs and sperm. These are called germline mutations and can be passed from parent to child: if a child inherits a germline mutation from parents, every cell in their body will have this error in DNA.
  Nonetheless, the speciation - distinguishing separate species - requires a diversification that concerns multiple aspects, including mating preferences and in a result is regarded to take millions of years. Therefore scientists assume that speciation is a process between two species the

original and the one which code changed. This assumption is a reason for limiting the representation of phylogenetic tree to binary tree. Multifurcating is used generally when it is hard to determine the sequence of taxa evolution.

As [89] claims, a remark made in [10] and [22] points the common misconception that some modern species are ancestral to other modern species. The proper interpretation should be that all modern species are found at leaves and one modern specie is as 'evolved' as any other. E.i. although mammals are thought to have evolved from something that resembled modern reptiles, modern reptiles are just as "old" evolutionarily as modern mammals.

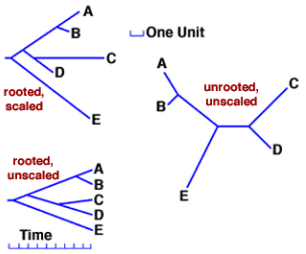


Figure 5.3: Various representations of a 5-taxa phylogenetic tree. Each of these trees represents the same five modern taxa: A, B, C, D, and E. The tree at upper left is rooted and scaled according to evolutionary distance. The root is at left. Taxa C and E have both undergone relatively large changes since divergence from the root, in contrast to taxa B and D. The tree at lower left is rooted and unscaled. Here the branch lengths are relative indicators of time since divergence. The tree at right is scaled but unrooted. In this tree, while the root is unknown, the relationships between taxa are identical to that shown in the other two trees. [96]

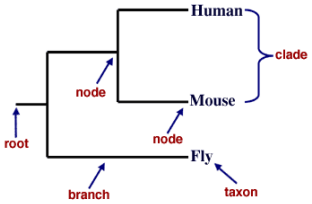


Figure 5.4: The parts of a phylogenetic tree. The taxa in this tree are "human", "mouse", and "fly" (all of which have had their full genomes sequenced). [96]

5.3 Phylogenetic tree's building material

Historical trees of life based on physical or morphological features (e.g. size, color, limbs). However, since genetics dominated in biology, information on the history of species is extracted from genetic material – mainly **nucleotids (DNA, RNA)** or **amino acids (proteins) sequences**.

The relationship between taxa is deduced from the **alignment** of several **sequences**, one from each examined taxa: the sequences are compared and with help of various algorithms, the similarity of sequences and sequences transformations are the basis for proposal of the history of spices evolution. When dealing with nucleotides, the alignment occurs usually when compared are identical nucleotide bases; in case of amino acids, alignment take place if two acids are identical or one can be derived from the other by substitutions that are likely to occur in nature.

The reasonable results are achieved by finding regions of similarity in homologous sequence which means the function of the compared parts of sequence is common or similar (e.i. a protein responsible for the same function of different taxa but with differences in construction). Phylogenesis take advantages of homologous sequences called orthologous, which indicate that the genes of different spices are similar or the same because they originated from a single gene of the last common ancestor.

The resemblance of DNA sequences taken from different organisms bases on theory that contemporary genetic material in all the living creatures has one common ancestral DNA. According to this theory, during the course of evolution, mutations occurred, creating differences between families of contemporary species.

Mutation is defined as a heritable change in the nucleotide sequence in the DNA, caused by a faulty operation process. These errors in replication occur often due to expose to ultra violet radiation or other environmental conditions. There are three kinds of gene mutations:

- substitution - a change of one nucleotide/protein in the DNA sequence.
- insertion - an addition of one or more nucleotides/proteins to the DNA sequence.
- deletion - a removal of one or more nucleotides/proteins from the DNA sequence.

Figure 5.5 shows several organisms (each in separate row) protein alignment

of a part of nucleotid sequence of a homologous sequence. By the same color there are marked regions of similarity.

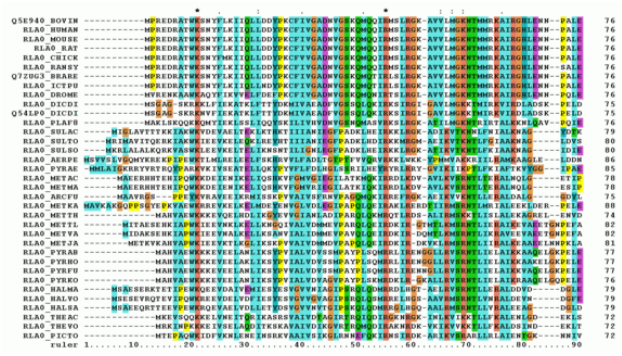


Figure 5.5: First 90 positions of a protein multiple sequence alignment of instances of the acidic ribosomal protein P0 (L10E) from several organisms. Generated with ClustalX. [97]

Mutations remain as a spice feature in two manners: deterministic and stochastic. Deterministic means by natural selection: mutations that are beneficial for spice's existence including the ones that evolved due to changes of environment. Stochastic are the ones that are random in the environment context. In phylogenesis, the more useful changes are the stochastic ones. It's because of the little probability that the same changes that are independent from the environment needs take place. The level of difference between different spices sequences is a base to determine how much they are related.

***

For further details on phylogeny and its application in computer science, consult [10], [16] or in Polish [32].

5.4 Phylogenetic tree's construction algorithms

Phylogenetics has developed various methods of building phylogenetic trees. All base on comparing homologous sequences trying to find **regions of similarity** and in the same time – **mutations** that occurred and caused differences in the sequences.

The methods can be classified according to different determinants such as, e.i. whether they consider probability models of sequences occurrences and

Table 5.1: Classification of phylogenetic tree's construction algorithms

Attribute characteristic	Algorithms distinguish different substrings in sequences and assign probability to occurrences of those substrings and to mutations that form those substrings, considering multiple mutations.	
Types	No models – each mutation and occurrence has the same probability.	Models of mutation and occurrence probability (basing on certain evolutionary assumptions).
Algorithms	MP	ML

Table 5.2: Classification of phylogenetic tree's construction algorithms.

Attribute characteristic	The requested tree can be obtained by building successively one tree or by generating a set of trees and seeking for the best.	
Types	Measuring the pair-wise distance/dissimilarity between two sequences, forming a resultant distance matrix and constructing the tree totally from the matrix.	Evaluating all possible trees and seeking for the one that optimizes the evolution.
Algorithms	NJ, UPGMA	MP, ML

mutations, whether they scan a number of possible trees solution and choose the most probable or build one result tree. In this chapter we briefly introduce three popular algorithms and in tables 5.1 and 5.2 we classify them according to some attributes.

MAXIMUM PARSIMONY (MP): The maximum parsimony method was one of the first reconstructing tree of life procedures. It scans all the possible trees, finding the one that with the lowest cost - the one in which the lowest number of mutations between the tree leaves had to occur. In order to do so, the method performs two assignments: a mutation to a branch and a state before the mutation (an ancestor) to an internal node incident to a branch. The total number of all mutations is the cost of a tree. The cheapest tree over all the possibilities is regarded as the best. The weakest part of the method is an assumption that all the mutations have the same probability (see table 5.1).

MAXIMUM LIKELIHOOD (ML): Maximal likelihood is a popular in statis-

tics method of finding the most probable values of requested parameters that generates known data – values that maximize the likelihood function. In phylogenetic case, the parameters are the tree topology and branches weights. Data is the set of sequences. The tree with the greatest likelihood function result is regarded to be the best. The major disadvantage of the method is its time complexity.

DISTANCES METHODS: Distance methods do not scan a number of probable trees, like the others two methods introduced in this chapter, but is a bottom-up clustering method for a tree creation, where a tree is built basing on a matrix d of distance between each pair of taxa (e.g. species or sequences). The disadvantage of those methods is that they give as a result only one tree, where there can be more trees with similar probability. Two popular distance methods are Unweighted Pair Group Method using Arithmetic averages (UPGMA) and neighbour joining (NJ). The former bases on the controversial hypothesis of the *Molecular clock* [23] which assumes that the rate of amino acid or nucleotide substitutions is approximately constant over evolutionary time. An algorithm of the latter, called in [33] *the most popular of distance methods*, is as follows:

The NEIGHBOUR JOINING algorithm begins with an unresolved star topology tree (fig 5.6a) and iterates over the following steps until the tree is completely resolved and all branch lengths are known [81]:

1. Based on the current distance matrix d, create a so-called Q-matrix as follows: for each pair i, j of on n leaves $Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(j,k)$
2. Find the pair of taxa f, g in Q with the lowest $Q(f,g)$value. Create a node u on the tree that joins these two taxa. (fig 5.6b)
3. Calculate the distance of each of the taxa in the pair to this new node. Those are the lengths of edges in the resultant tree to the just created internal node: $d(f,u) = \frac{1}{2}d(f,g) + \frac{1}{2(n-2)}\left[\sum_{k=1}^{n} d(f,k) - \sum_{k=1}^{n} d(g,k)\right]$ and, by reflection $d(g,u) = d(f,g) - d(f,u)$
4. Calculate the branch distance of each taxon k outside of this pair to the new node as follows: $d(u,k) = \frac{1}{2}[d(f,k) + d(g,k) - d(f,g)]$. Go back to step 1, considering a single taxon u instead of the pair of joined neighbours.

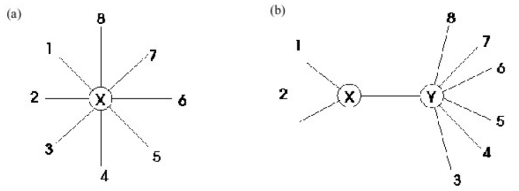


Figure 5.6: NJ method: (a) an unresolved star-like tree (b) the closest terminals(for definition, see text) 1 and 2 are joined and a branch is inserted between them and the remainder of the tree and the value of the branch is taken to be the mean of the two original values. This process is continued until only one terminal is left. [87]

Chapter 6

How the Tree of Life Impacts Our Life

Phylogeny is mainly applied in pure science-oriented domains. However, the tree of life researches have also impressed its value in other fields. In this chapter we present the results of researches in some fields that phylogeny have impact on.

SCIENCE
Phylogenetic trees are used to classify relationship between species and to reconstruct their evolution. Basing on phylogenetic tree, scientist observe the origins and further evolution history of species. They search for adaptation syndromes that form simultaneously with changes in the environment, they investigate if the new features forming are related to sudden spreading over of the organisms groups.

Among other issues, the focus of interest is put to the genealogical relation between human and animals. Before the biologists' molecular researches, palaeontologists studying fossils,had already concluded (prior to 1960) that chimpanzees and gorillas are human's closest relatives (fig. 6.1). But that the relationship was distant: the split, leading to humans on the one hand and chimpanzees and gorillas on the other, having occurred some 15 million years ago. The detailed molecular data, obtained by immunological studies in the 1960s [11] confirmed that humans, chimpanzees and gorillas indeed are closely related, but suggested that the relationship is much closer, indicating that this split occurred only 5 million years ago. The latter theory is currently accepted.

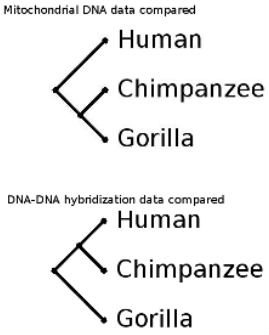


Figure 6.1: Evolutionary relationship between humans, chimpanzees and gorillas is ambiguous. The reason for these conflicting results is the close similarity between DNA sequences in the three species, the differences being less than 3% for even the most divergent regions of the genomes.

MEDICINE

Phylogeny bases on molecular researches just like modern medicine. Searching for taxa similarities and mutations has naturally proved to be helpful in e.i. epidemiology. Observing the evolution process of viruses can help to diagnose its related kinds, investigate which virus' sequence parts are responsible for which infected human's reactions and eventually – how to cure.

Currently, one of the areas where phylogeny provides significant help if the HIV virus problem. Acquired immune deficiency syndrome (AIDS) is regarded as the global epidemic. The illness is caused by a human immunodeficiency virus 1 (HIV 1) – a virus that infects cells involved in the immune response.

Unlike human DNA, which remains stable for a lifetime, HIV's RNA changes very rapidly, which leads to a huge amount of genetic diversity. Using phylognetic analysis, this diversity allows to ascertain where HIV comes from as well as track various strains of HIV that exists worldwide.

Epidemiologists began an investigation of the historic and social conditions that might have been responsible for the start of the AIDS epidemic. Speculation centred around the discovery that similar immunodeficiency viruses are present in primates such as the chimpanzee, sooty mangabey, mandrill and various monkeys. These simian immunodeficiency viruses (SIV) are not pathogenic in their normal hosts but it was thought that if one had become transferred to humans then within this new species the virus might have acquired new properties, such as the ability to cause disease and to spread

rapidly through the population.

Comparison between virus DNA sequences has resulted in the reconstructed tree shown in figure 6.2. A star-like pattern with different samples of HIV-1 (the type of HIV that causes the epidemic) that radiates from one end of the unrooted tree (the bottom of the figure) shows that they have a slightly different sequences. Scientists conclude this topology implies that the global AIDS epidemic began with a very small number of viruses, perhaps just one, which have spread and diversified since entering the human population.

The closest relative to HIV-1 among other primates is the SIV of chimpanzees. This leaded to the conclusion that the virus jumped across the species barrier between chimps and humans causing the AIDS epidemic. However, this epidemic did not begin immediately. A relatively long uninterrupted branch links the center of the HIV-1 radiation with the internal node leading to the relevant SIV sequence. This suggests that after transmission to humans, HIV-1 underwent a latent period and remained restricted to a small part of the global human population, presumably in Africa, before beginning its rapid spread to other parts of the world.

Moreover, there are discovered more AIDS-causing viruses, related to HIV-1, not yet causing global epidemic, e.i. HIV-2 or ZR59. It appears that HIV-2 was transferred to the human population independently of HIV-1, and from a different simian host.

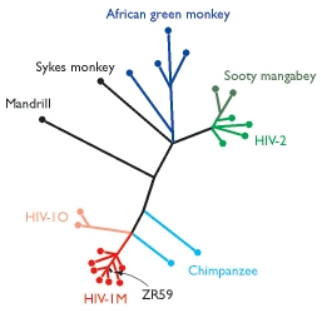


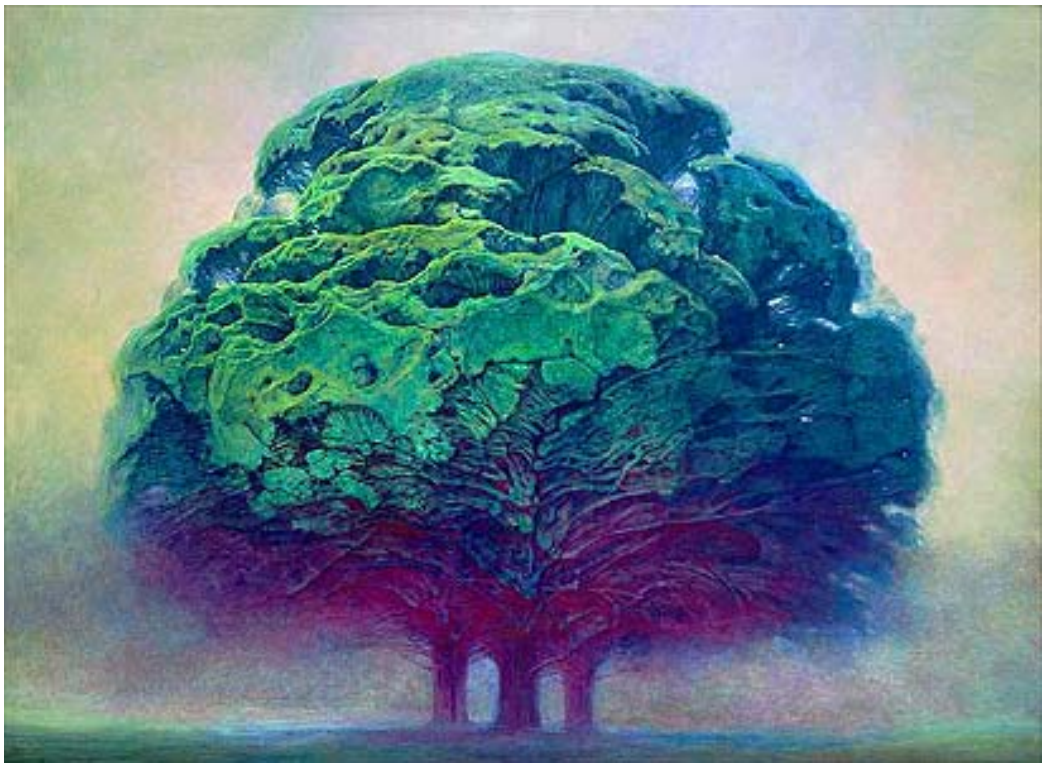
Figure 6.2: The star-like phylogenetic pattern formed by genomes related to SIV. AIDS epidemic is due to the HIV-1M type of immunodeficiency virus. [11]

LAW
In the past decade, phylogenetic analyses have played a significant role in successful convictions in several criminal court cases. The [89] author refers to [1], [28], [2] and 8 more papers of which most refer fo the the USA court

cases where HIV virus phylogeny was a proof. Matthew Weait from the University of London in his paper [93] outlines a brief history of phylogenetic analysis being legally admissible evidence in investigations and court. The first analysis was used in a court of law in Sweden in 1992 and beforehand in 1990 and 1991 it was helpful in United States Centres for Disease Control investigation. In all the cases the analysis were to investigate whether an accused was the source of HIV infection. Despite sexual contacts, in the paper there are also refereed medical negligences. More USA court cases from the late 90's and cases from other countries (Australia, Denmark, Belgium) from this century are mentioned, signalizing that application of phylogenetics to court cases is not an isolated case.

Part II

Phylogenetic Trees Distances Algorithms



Beksiński, 1986, [100]

The affinities of all the beings of the same class have sometimes been represented by a great tree... As buds give rise by growth to fresh buds, and these if vigorous, branch out and overtop on all sides many a feebler branch, so by generation I believe it has been with the great Tree of Life, which fills with its dead and broken branches the crust of the earth, and covers the surface with its ever branching and beautiful ramifications.

Charles Darwin, 1859

Introduction

Although it is assumed that the current spices evolution is describable with one phylogenetic tree, the diverse phylogenetic methods or different choices of input genes sequences lead to a variety of phylogenetic trees outputs. The need to acquire a consensus tree and also to compare results and to rate phylogenetic algorithms leaded to a meta-calculations of comparing phylogenetic trees.

In this part II of the paper, we present an outline of methods of phylogenetic tree distance definition - presenting the general steps that constitute a template for trees metrics, the classical metrics (chapter 7) and the new approach to measuring phylogenetic trees distances - the Minimum Weight Perfect Matching metrics (chapter 8).

Chapter 7

Classical Metrics

In this chapter we present the classical, most popular phylogenetic trees metrics. There have been released a numerous papers on their idea and developed algorithms of progressive efficiency (e.i. for quartet distance, among others: [14], [15], [9], [34]) and some of them are applied in in phylogenetic libraries (see chapter 9). In this paper we present the popular metrics on phylogenetic trees: their definition, brief description of naive algorithm and more sophisticated, more efficient algorithms that were implemented in the *PhylotreeDist* library.

Before presenting the concrete algorithms that counts how much do trees differ, we treat the trees distances generally, describing the universal method of computing phylogenetic trees distances and classifying the algorithms described in this paper into categories.

7.1 The model of phylogenetic trees metric definition

Computing distances among trees is generally a schematic procedure: the algorithms (at least the most popular ones) operate on just two trees and they can be modelled with the following expression:

$$d(T_1, T_2) = g_h(f(T_1), f(T_2)) \tag{7.1}$$

where

- $f(T)$ is a function that extracts a set of *description elements* from the tree T. A description element is a piece of information on tree's topology, mainly leaves relations; a tree can be reconstructed from the set of all its description elements.
- h defines how to compare a description element from one tree to the other tree's description elements set (one or more description element from the other tree).
- g_h is a metric that computes the total distance basing on h definition.

According to equation 7.1, the proceedings for computing a distance between phylogenetic trees is to:

1. For each tree, create a set of its description elements according to f.
2. Computes the total distance so that every description element in each tree is considered in a total result. The elements are compared according to h definition.
   To reduce the time complexity, the distances between description elements are often computed separately from the total distance computing.

We now categorize all the distances between phylogenetic trees algorithms that are described in in the part II of this paper, according to the algorithm's g_h, h, f definitions. The assignment of algorithms to the categories is presented in table 7.1.

$f(T)$ **- a type of description element:**

a. leaves partitioning – a description element describes which of all the tree leaves belong to a partition. Sometimes the relations among the leaves in the partition also have importance (e.i. in pairs minimum weight perfect matching metric). The number of description elements depends on the number of internal nodes or internal edges. The tree can be reconstructed from the set of all its partitions.

b. leaves k-sets topology - a description element describes the relation, topology of k leaves. There are $\binom{n}{k}$ description elements for every tree (n – the number of leaves). The tree can be reconstructed from the set of all its k-sets.

h - a method of comparing $f(T_1)$ to $f(T_2)$:

a. checking if a description element exists in other tree's elements set. The distance is 0 if *yes* and 1 if *no*.

b. measuring how much different is one description element from another. The result is an absolute value.

g_h - computing the total distance:

a. a p-norm on the space of distances between description elements defined by h

$$g_{h,p} = ||h(f(T_1), f(T_2))||_p \tag{7.2}$$

b. a perfect matching on a complete bipartition graph where each partition is one of the two trees' description elements set, nodes are the description elements and edges are weighted with the distance from a one tree's description element to another tree's description element that is computed according to h.

category\function	$f(T)$		h		g_h	
a	leaves partitioning	rf,mp, ms,mc	check existence	rf, q,t,	p-norm	rf,q t,n
b	leaves k-sets	q,t n	check difference	n,ms, mc,mp	perfect matching	ms,mc mp

Table 7.1: The classification of classical and perfect matching algorithms. rf - Robinson-Foulds, q - quartets, t - triplets, n - nodal, ms - MWPM splits, mc - MWPM clusters, mp - MWPM pairs (MWPM - minimum weight perfect matching)

7.2 Robinson-Foulds distance

The distance between two trees, called after names of its claimants the Robinson-Foulds (RF) metric is regarded as a standard measure used in Phylogenesis [12]. The researches on which tree distances algorithms are used in phylogenetic libraries in chapter 9 show that Robinson-Foulds metric is the standard (and often the only one) metric provided by the libraries.

D.R. Robinson and L. R. Foulds describe in their (released in 1981) paper [29] a way to compare two multifurcating trees with the same leaves sets. The metric is defined as a count of the symmetric differences between sets of either clusters (rooted case) or bipartitions (unrooted case) of the two trees.

TAGS
cluster, bipartition, internal node, internal edge, consensus tree, strict consensus tree

INPUT DATA
Two rooted or two unrooted phylogenetic trees $T_1, T_2 \in T_L$ with the same set of leaves L.

DEFINITION
The Robinson-Foulds distance between two trees T_1 and T_2 is defined as follows:

For unrooted trees:

$$d_{RF}(T_1, T_2) = |\beta(T_1) \oplus \beta(T_2)|$$

where $\beta(T)$ is a $|E| - |L|$ size set of nontrivial bipartitions at a tree.

For rooted trees:

$$d_{RF}(T_1, T_2) = |\sigma(T_1) \oplus \sigma(T_2)|$$

where $\sigma(T)$ is a $|V| - |L|$ size set of nontrivial clusters at a tree.

Briefly, the distance is a symmetric difference between either clusters or bipartitions of two trees.

Table 7.2 shows this definition in the nomenclature from section 7.1.

function	description
$f(T)$	returns a set of all T's nontrivial bipartitions\clusters
h	returns 1 if a description element of one tree does not exist in the other tree, 0 otherwise
g_h	$g_{h,1} = \sum_{b \in f(T_1)} h(b, f(T_2)) + \sum_{b \in f(T_2)} h(b, f(T_1))$

Table 7.2: Definition of the Robinson-Foulds distance in the nomenclature from section 7.1.

THE GENERAL ALGORITHM AND ITS TIME COMPLEXITY
The general algorithm performs two actions:

1. Collecting all description elements of each tree. The way the description element is constructed depends on whether it is rooted or not:

   - **Unrooted trees:** The algorithm inspects every tree's internal edge. A removal of an edge splits the tree into two subtrees: the algorithm assigns to one set a list of all leaves of one of the subtrees and the remaining leaves to a second set, creating a *bipartition* also called a *split*. There are at most $|L| - 3$ internal edges in a tree topology graph which is the number of bipartitions.
   - **Rooted trees:** The algorithm inspects every tree's internal node. The node is always a root to some subtree (the node *induces* a subtree). The leaves of subtree form a *cluster*. There are at most $|L| - 1$ internal nodes that induce clusters.

   Both types of description element - cluster and bipartition - divide leaves into two categories, the difference is that for bipartition each category has equal meaning, whereas for cluster only the leaves that are members of a considered cluster have a matter.

   The recursive algorithm performs creating description elements in $O(|L|^2)$ time as for each description element there is performed an iteration through $|L|$ leaves list in order to mark them as belonging or no to the partition.

2. Symmetric difference on the T_1 and T_2 sets: for every T_1's and every T_2's bipartition/cluster – the algorithm exams whether the element is also a member of the other tree and sums up the number of negative answers.

   As there are $O(|L|)$ operands: bipartitions/clusters, there are $O(|L|^2)$ operations performed on the operand which size depends on $|L|$ - which gives an overall $O(|L|^3)$ time. In practice, the membership to a bipartition is represented as a record with binary flags where each flag represents a leaf.

However, in 1985 William H.E. Day presented in [19] the algorithm which time and space complexity is linear. Below we present the algorithm.

THE $O(n)$ DAY's SOLUTION
The Robinson-Foulds distance between two trees and is measured as:

$$d_{RF}(T_1, T_2) = M(T_1) + M(T_2) - 2 \cdot M(T_{cons}) \tag{7.3}$$

where:

$M(T)$ - a number of internal nodes in T
T_{cons} - a *strict consensus tree* - a tree that groups only clusters/bipartitions that appear in all considered trees.

The equation 7.3 bases on three notices:

- (1) a symmetric difference of cluster sets for trees T_1 and T_2 fulfils $|\sigma(T_1) \oplus \sigma(T_2)| = |\sigma(T_1)| + |\sigma(T_2)| - |\sigma(T_1) \cap \sigma(T_2)|$,
- (2) $\sigma(T_{cons}) = |\sigma(T_1) \cap \sigma(T_2)|$,
- (3) every cluster is associated with an internal node (a root of a cluster).

Basing on the above, W.Day presents in his paper [19] an algorithm to build a consensus tree in the linear time. The crucial element of this algorithm are the specific structures. They require that:

- All the input trees nodes (internal and leaves) are distinctly labelled.
- Best way to label leaves is to give them numbers $[0..(|L| - 1)]$ (as they will be indices of an array).
- The leaves in a subtree generated by an internal vertex must be ordered in the plane from left to right.

The algorithm operates on a rooted tree, but Day in his paper defines a reduction function that modifies unrooted trees into rooted ones by deleting from an unrooted tree one of the leaves and the incident edge, then rooting the tree at the internal node the deleted edge was adjacent to.

Here we present the steps taken to acquire the $O(n)$ (n – the number of all trees nodes, $n \leq 2|L| - 1$) Robinson-Foulds distance result. We drive special attention to the structures used in the algorithm.

1. For each tree, the algorithm creates a postorder sequence of its nodes - called a *PostorderTree* or a *PSW* - Postorder Sequence with Weights.

   The structure's construction is following: during a DFS search of a tree, a node x on which searching is finished (there is no more child to examine) obtains a successive position in a PSW array and this position is its $pswId$. Under this position, the array stores two data about the node: node's original label (v_x) and the number of node's descendants (s_x) called in Day's paper a "weight".

   The structure allows to scan the tree in a way that all the leaves of each cluster are examined first, and then the internal x node that is the root

of the cluster is examined. Its s_x value allows to determine how many of preceding elements in a PSW array are the cluster's elements (leaves and internal nodes).

The 7.3 table and its description complements this definition. For the sample see tables 7.6 and 7.7.

The time required to execute this step is the DFS algorithm time which is $O(n)$.

PSW's index	fields	
postorder position *pswId*	vertex label v_x	number of descendants s_x
0	$v_i \in \{0, 1, .., (n-1)\}$	$s_i \in \{0, 1, .., (n-1)\}$
1	$v_j \in \{0, 1, .., (n-1)\}$	$s_j \in \{0, 1, .., (n-1)\}$
..	...	...
$n-1$	$v_k \in \{0, 1, .., (n-1)\}$	$s_k \in \{0, 1, .., (n-1)\}$
	note: $v_i \neq v_j \neq ... \neq v_k$	

Table 7.3: The postorder sequence structure PSW built on a tree T with n nodes. The structure is an array that contains n elements. The index of an array is its postorder index *PSWid*. Each array element consists of two items: the unique label of T's node and the number of nodes that are the descendants of the considered node; if a considered node is a leaf, this value is 0.

2. For one arbitrary chosen tree (later referenced as the *base tree*), there is constructed a cluster structure $ClustersStructure$ (called in H.Day's paper the $ClusterTable$) that using certain mapping, stores the information on every tree's cluster.

   The $ClustersStructure$ functionality is that while examining the remaining tree (later referenced as the *examined tree*), the structure updates information on the clusters: for each cluster in the examined tree - the algorithm checks whether the cluster is in the $ClustersStructure$. Finally the cluster table stores just the clusters of the resultant consensus tree.

   To construct the structure there is performed a traversal of base tree's PSW. The structure's construction, presented in tables 7.4 and 7.5 is following: it contains two arrays - the mapping one $mappingArray$ that assigns a leaf's label to its *postorder leaf position* and the $clustersArray$ that stores information on which base tree's clusters are also present in the other tree.

- The *mappingArray* (table 7.4, example in table 7.8) takes advantage of the fact that leaves labels are the successive numbers from 0 to $|L| - 1$ and treats the labels as its indices. When the PSW traversal encounters a leaf, it increments a special leaf index that indicates the number of already scanned leaves. This index is called a *postorder leaf index* and is a hash value for the leaf label in the *mappingArray*.
- The *clustersArray* has as many elements as the number of leaves. Its indices are the postorder leaf indices *psLid*. An element stores:
  - a representation of a cluster: an ordered pair (L, R) of the respectively leftmost and rightmost cluster's leaf in the sense of a postorder leaf position in the base tree.
  - a validation flag: flags assigned to each cluster that marks if a cluster exists in a consensus tree

  The position of the element is either the leftmost or the rightmost leaf of the cluster so either L or R value of the cluster representation element. The choice is done according to the following hashing rule: the position of the leftmost leaf is chosen if the next node after the cluster's root (in the PSW sense) is not a leaf (so if its subtree size s value is not 0). Otherwise the position of the rightmost leaf is chosen (so if the next node is a leaf). The last cluster examined – the one rooted at a the root of the tree root - is stored in its rightmost leaf position.
  The rule guarantees that each cluster has got its position in the container.

The time complexity of constructing the *ClustersStructure* is the $O(n)$ PSW tree traversal time.

key	mapping
leaf label v_x	base tree's postorder leaf index $psLid_x$
0	$psLid_0 \in \{0, 1, .., (\|L\| - 1)\}$
1	$psLid_1 \in \{0, 1, .., (\|L\| - 1)\}$
...	...
$\|L\| - 1$	$psLid_{\|L\|-1} \in \{0, 1, .., (\|L\| - 1)\}$
	note: $psLid_0 \neq psLid_1 \neq ... \neq psLid_{\|L\|-1}$

Table 7.4: The structure of *mappingArray* from the *ClustersStructure*

clustersArray index	fields	
base tree's postorder leaf index *psLid*	pair of cluster's leftmost and rightmost psLid (L, R)	validation flag f
0	(L_0, R_0)	$f_0 \in \{true, false\}$
1	(L_1, R_1)	$f_1 \in \{true, false\}$
..	...	...
$\|L\| - 1$	(L_n, R_n)	$f_{\|L\|-1} \in \{true, false\}$
	note: $(L_i, R_i) \in$ $\{\emptyset, (psLid_i, psLid_x), (psLid_y, psLid_i)\}$, $x, y \in \{0, 1, \|L\| - 1\}$; there are $n - \|L\|$ pairs $(L_i, R_i) \neq \emptyset$	

Table 7.5: The structure of *clustersArray* from the *ClustersStructure*

3. The examined PSW tree is traversed, during which the clusters in the *clustersArray* are marked valid if they have the same leaves sets as clusters in the examined tree.

   The crux of the $O(n)$ algorithm that allows to compare all the clusters of both trees in linear time lies in the following procedure that uses a stack of maximally n elements:
   For each node visited during the examined PSW tree traversal, if it is

   - a leaf - we push on the stack its number of descendants $s_x = 0$ and its *psLid*.
   - an internal node x - we firstly pop some number of nodes from a stack: we count a popped leaf as one descendant, we count a popped internal node y as one descendant plus its number of descendants s_y. We perform popping the nodes until the count is equal to s_x. We then check whether a cluster induced by x exists in the *clustersArray* (see description below) and finally we push information about x on the stack: s_x, the smallest and the greatest *psLidx* among cluster's the leaves.

   In this way during its lifetime on the stack there is n elements pushed/poped - therefore the time of the examined tree traversal is $O(n)$.

   Checking whether a cluster induced by an internal node x exists in the *clustersArray* is as follows: after popping the nodes, we determine which of the cluster leaves' *psLid* index is the lowest-l and which the greatest-r. If both:

- the number of leaves in the cluster is equal to $r - l + 1$
- under either of l, r positions in the $clustersArray$, $L = l$ and $R = r$)

conditions are fulfilled, then the examined cluster is the same as this represented by (L, R) cluster from $clustersArray$.

The above is true due to the fact that if the number of leaves in the examined cluster is equal to $r - l + 1$, then the $psLidx$s of its leaves have to be successive values from l to r, as each value is different and as l and r are boundary. Simultaneously, in the $clustersArray$, (L, R) is the representation of a cluster of $L - R + 1$ leaves with postorder leaf indices from L to R.

The above description is briefly presented in the 1 pseudocode:

Algorithm 1 Robinson-Foulds linear time complexity algorithm

```
1: integer function LinearRobinsonFoulds(T1, T2)
2: if T1 and T2 are unrooted then
3:    T1.Root();
4:    T2.Root();
5: end if
6:
7: PostorderTree pswT1(T1);
8: PostorderTree pswT2(T2);
9: ClusterStructure c(pswT1);
10: c.RemoveUncommonClusters(pswT2);
11: return T1.GetNumberOfInternalNodes() +
    T1.getNumberOfInternalNodes() – 2 × ct.GetNumberOfInternalNodes();
```

SAMPLE

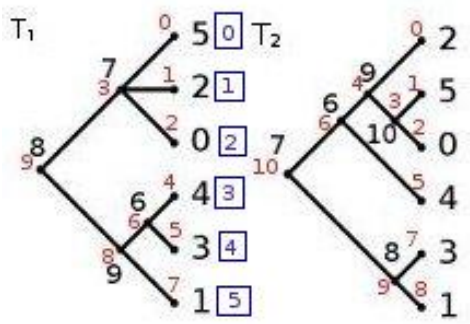


Figure 7.1: Two rooted trees T_1 and T_2. The bigger index numbers are the original labels of a tree. The smaller are the postorder tree indices $pswId$. The values in the squares are T_1's postorder leaves indices $psLid$.

$pwsT_1$'s index	vertex label	number of descendants
0	5	0
1	2	0
2	0	0
3	7	3
4	4	0
5	3	0
6	6	2
7	1	0
8	9	4
9	8	9

Table 7.6: Postorder sequence (PSW) of tree T_1 from figure 7.1

The figure 7.1 shows two input sample trees T_1 and T_2. They have each node labelled (the bigger size numbers). For the convenience of the algorithm, the leaves labels are sequential numbers $[0..|L-1|]$.

Firstly, while performing a DFS search on T_1 and afterwards T_2, the postorder sequence trees $pswT_1$ and $pswT_2$ are built: to each node there is assigned a postorder sequence index $pswId$ (apparent in figure 7.1 as the smaller labels) and a a number of descendants - the number of vertices in the subtree rooted

$pwsT_2$'s index	vertex label	number of descendants
0	2	0
1	5	0
2	0	0
3	10	2
4	9	4
5	4	0
6	6	6
7	3	0
8	1	0
9	8	2
10	7	10

Table 7.7: Postorder sequence (PSW) of tree T_2 from figure 7.1

leaf label	base tree's postorder leaf index
0	2
1	5
2	1
3	4
4	3
5	0

Table 7.8: The *mappingArray* - one of the two containers that form a *ClustersStructure*. The *mappingArray* assigns a postorder leaf position to a leaf.

base tree's postorder leaf index (psLid)	cluster's leftmost psLid	cluster's rightmost psLid	is valid? - flag
0			
1			
2	0	2	T
3	3	5	
4	3	4	
5	0	5	T

Table 7.9: The *clustersArray* - one of the two containers that form a *ClustersStructure*. It assigns all consensus tree's clusters to their either leftmost or rightmost (in the postorder leaf position sense) leaf. The validation flag marks the clusters that are also found in the tree that is examined.

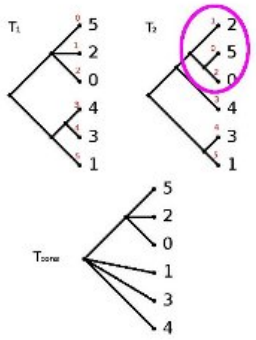


Figure 7.2: Two trees T_1 and T_2 and the strict consensus of them $Tcons$. The circled cluster from T_2 is the only one, besides the one containing all the trees, that is common for T_1 and T_2.

at examined node excluding the node itself. For the figure 7.1, the values are presented in tables 7.6 and 7.7.

Next, $pswT_1$ becomes a base of a consensus tree which clusters are stored in the $ClustersStructure$: during a postorder traversal of $pswT_1$, the $mappingArray$ (7.8) and the $clustersArray$ (7.9) are filled.

Afterwards, during the postorder $pswT_2$ traversal, the clusters from T_2 are compared to the data in cluster table ($ct1$). In the example, the only T_2 internal node that builds a subtree containing the same leaves as any of T_1 subtrees is, beside the root, the one with original id 9 (the cluster is circled in the figure 7.2). Therefore the value of the Robinson-Foulds distance between trees T_1 and T_2 is $d_{RF}(T_1, T_2) = 4 + 5 - 2 \times 2 = 5$.

7.2.1 Weighted Robinson-Foulds

If the input trees are unrooted and have weighted edges, we can compute the Robinson-Foulds distance with regard to those weights. In this case each split has a weight equal to the weight of a an edge which removal induces the split. This also concerns trivial splits, that cannot be omitted in the algorithm. The distance is given by a following expression:

$$d_{wRF}(T_1, T_2) = \sum_{b \in \beta'(T_1) \setminus \beta'(T_2)} w_1(b) + \sum_{b \in \beta'(T_2) \setminus \beta'(T_1)} w_2(b) + \sum_{b \in \beta'(T_1) \cap \beta'(T_2)} |w_1(b) - w_2(b)|$$

where $\beta'(T_i)$ is an $|E|$-sized set of all the bipartitions of a tree T_i and $w_i(b)$ is weight of an edge of T_i which removal leads to a bipartition b.

Notice that if every edge in both trees has weight 1, then the weighted RF distance between the trees is equal to classical RF distance which does not consider edges' weights.

7.3 Quartet distance

In the described in the previous section Robinson-Foulds algorithm, each description element divided all the leaves into two groups and the distance was acquired by counting how many of description elements appear in only one of the tree. The metrics described in this and next section have the same method of comparing the description elements, but the elements themselves are topologies of k-leaves subsets of a tree, where k is in the quartet distance case equal to 4 and in the triplets distance case - equal to three.

TAGS
quartet, butterfly topology of a quartet, unrooted tree

INPUT DATA
Two unrooted phylogenetic trees $T_1, T_2 \in T_L$ with the same set of leaves L.

DEFINITION
A *quartet* is a set of four species; a *quartet topology* of a set $a, b, c, d \in S$ of four species is the topological subtree of T induced by these species. The four possible quartet topologies for species a, b, c, d are shown in figure 7.3. [9]
Given two evolutionary trees on the same set of n species, the quartet distance between them is the number of sets of four species for which the quartet topologies differ in the two trees.

Table 7.10 shows this definition in the nomenclature from section 7.1.

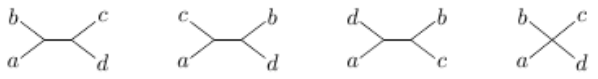


Figure 7.3: The four possible quartet topologies of species a, b, c, and d. [9]

THE ALGORITHM
As for an n-leaves tree there are $\binom{n}{4}$ sets of four species. The quartet distance can be calculated in time $O(n^4)$ by examining the sets one by one. As it is

function	description
$f(T)$	returns a set of all T's quartets
h	returns 1 if a quartet of one tree has different topology in the other tree, 0 otherwise
g_h	$g_{h,1}(f(T_1), f(T_2)) = \sum_{q \in f(T_1)} h(q, f(T_2)) + \sum_{q \in f(T_2)} h(q, f(T_1))$

Table 7.10: Definition of the quartets distance in the nomenclature from section 7.1.

not very efficient, there were developed other algorithms and currently the most efficient are the $O(n \log n)$ time complex one that works only on binary trees (for the details, see [9]) and for multifurcating trees $T_1 = (V_1, E_1), T_2 = (V_2, E_2)$ the $O(|L| + |V_1 \setminus L|\ |V_2 \setminus L|\ min\{\Delta(T_1 \setminus L), \Delta(T_2 \setminus L)\})$ time, $O(|L| + |V_1 \setminus L|\ |V_2 \setminus L|)$ space complex. The latter algorithm is briefly outlined below, for details consult [15].

The method bases on finding in the two input trees the butterfly topology quartets that are and are not shared by the trees. A *butterfly topology quartet* is a topology of four leaves where one pair of them is separated from the other pair by an edge. Out of four possible quartet topologies there is only one that is no butterfly, in the figure 7.3 it is the one on the right. We say that a butterfly quartet is *shared*, if it has the same butterfly topology in both trees. Otherwise, we say that the butterfly quartet is *nonshared*.

In [15] Christiansen et al. argue that the total quartet distance of a tree with arbitrary degree can be calculated considering only butterflies quartets in a following way:

$$\begin{aligned} qdist(T_1, T_2) = \quad & shared(T_1, T_1) + shared(T_2, T_2) \\ & -2shared(T_1, T_2) - nonshared(T_1, T_2) \end{aligned} \tag{7.4}$$

where $qdist(T_1, T_2)$ is the quartet distance value, $shared(T_1, T_2)$ denotes the number of quartets that are butterflies with the same topology in tree T_1 and tree T_2 and $nonshared(T_1, T_2)$ denotes the number of quartets that are butterflies in both T_1 and T_2 but with different topology.

The [15] proves the equation as follows: let Q_1 denote the number of quartets which have butterfly topology in T_1 and non-butterfly topology in T_2. Symmetrically, let Q_2 denote the number of quartets which have butterfly topology in T_2 and non-butterfly topology in T_1. A butterfly quartet in T_1 is either a butterfly quartet in T_2 or a non-butterfly quartet in T_2.

The number of butterfly quartets in T_1 , $shared(T_1, T_1)$, can thus be expressed as the sum $shared(T_1, T_2) + nonshared(T_1, T_2) + Q_1$. Similarly, the sum $shared(T_2, T_2) = Q_2 + shared(T_1, T_2) + nonshared(T_1, T_2)$. It is now straightforward to verify that the righthand side of equation 7.5 is $Q_1 + Q_2 + nonshared(T_1, T_2)$ so is the number of quartets where the quartet topologies differ in T_1 and T_2 - the $qdist(T_1, T_2)$.

For brevity, we denote id_i to be an *internal degree* - a maximal number of non-leaf nodes adjacent to a non-leaf node in T_i; therefore $id_i = \Delta(T_1 \setminus L)$. Cristiansen et al. argue that (7.5) can be computed in the $O(|L| + |V_1 \setminus L||V_2 \setminus L|min\{id_1, id_2\})$ time. More precisely - it needs $O(|L| + |V_1 \setminus L||V_2 \setminus L|)$ time for a preprocessing step, $O(|V_1 \setminus L||V_2 \setminus L|)$ for calculating $shared(T_1, T_2)$, $O(|V_1 \setminus L||V_2 \setminus L|min\{id_1, id_2\})$ for calculationg $nonshared(T_1, T_2)$ and $O(max(|V_1 \setminus L|, |V_2 \setminus L|)$ for calculating $shared(T_i, T_i)$.

The algorithm bases on scanning subtrees connected to internal nodes: Let F_i, $i \in \{1..deg(v)\}$ be a set of leaves of a subtree connected to internal node v e.g. all the leaves in a tree rooted at a node adjacent to v (see figure 7.4) and lets $\bar{F}_i = L \setminus F_i$. We denote the size of F_i by $|F_i|$ and the size of $\bar{F}_i$ by $|\bar{F}_i|$.

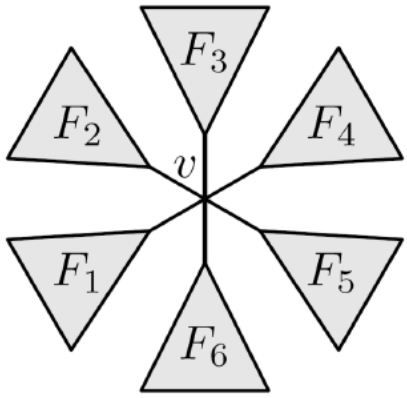


Figure 7.4: Six subtrees connected to the internal vertex v. Source [15].

We say that v *claims* a butterfly quartet $ab|cd$ if $a, b \in F_i, c \in F_j, d \in F_k$, where $i \neq j \neq k$ and $i, j, k \in \{1..deg(v)\}$, see figure 7.5. With this definition, each butterfly quartet is claimed by exactly two internal nodes (e.i. for quartet $ab|cd$, one internal node claims that a, b is in one subtree, c is in separate, d in is separate subtree and an other internal node claims that c, d is in one subtree, a is in separate and b is in separate subtree).

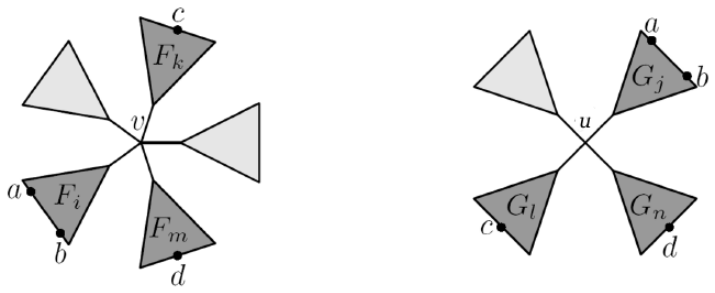


Figure 7.5: Internal nodes $v \in T_1$ and $u \in T_2$, each claiming the quartet $ab|cd$. Source [15].

Given F - a set of leaves of T_1's subtree and G - a a set of leaves of T_2's subtree, we call the *shared leaf set* an intersection $F \cap G$ e.g. the set of leaves present in both F and G. Its size is denoted by $|F \cap G|$. $|F \cap G|, |\bar{F} \cap G|, |F \cap \bar{G}|$ and $|\bar{F} \cap \bar{G}|$ are also sizes of shared leaf sets between a single subtree from T_1 and a single subtree from T_2.

The proposed algorithm bases on dynamic programming and assumes that the sizes of the intersections and the number of leaves in all the subtrees are computed beforehand and access to them is $O(1)$. The latter can be done in $O(|L|)$ time simply traversing the tree. The former can be obtained also using trees traversal in $O(|L| + |V_1 \setminus L||V_2 \setminus L|)$ time. The $|\bar{F} \cap G|, |F \cap \bar{G}|$ and $|\bar{F} \cap \bar{G}|$ values can be computed using the following equations:

$$|\bar{F} \cap G| = |G| - |F \cap G|$$

$$|F \cap \bar{G}| = |G| - |F \cap G|$$

$$|\bar{F} \cap \bar{G}| = n - |F| - |G| + |F \cap G|$$

shared(T, T)
In the 7.5 equation, the number of butterfly quartets in a single tree, $shared(T, T)$, can be computed with

$$shared(T, T) = \frac{1}{2} \sum_{v \in V(T) \setminus L} \sum_{i \in \{1..deg(v)\}} \binom{F_i}{2} \left(\binom{\bar{F}_i}{2} - \sum_{i \neq j} \binom{F_j}{2} \right) \tag{7.5}$$

as for each internal vertex v there is performed a count of occurrences when a pair of leaves is in one subtree and the remaining two leaves are in the set built of all the leaves that are not in that subtree and in the same time those remaining two leaves are not together in any other subtree. With

use of dynamic programming, the time complexity is $O(|V \setminus L|)$; for details consult [15].

shared(T_1, T_2)
The number of butterfly quartets shared by trees T_1, T_2, $shared(T_1, T_2)$ is the sum over each pair of internal nodes $v \in V_1$, $u \in V_2$ of $shared(v, u)$ which is the number of shared butterfly quartets claimed by both internal nodes. Assume that $F_1, ..., F_{deg(v)}$ are sets of leaves in subtrees of v and $G_1, ..., G_{deg(u)}$ are sets of leaves in subtrees of u. We wish to count all quartets $ab|cd$ where $a, b \in F_i \cap G_j, c \in F_k \cap G_l$ and $d \in F_m \cap G_n, i \neq k \neq m, j \neq l \neq n$ (see figure 7.5). Counting the possible combinations of a and b is expressed by the following double sum, which for internal nodes v and u sums over all pairs of subtrees leaves sets :

$$\sum_i \sum_j \binom{|F_i \cap G_j|}{2}$$

Given that a and b are in $F_i \cap G_j$, we need to find c and d in $\bar{F}_i \cap \bar{G}_j$. The number of possible choices of c and d is expressed by:

$$\binom{|\bar{F}_i \cap \bar{G}_j|}{2}$$

.

However when finding c and d in $\bar{F}_i \cap \bar{G}_j$, the condition that c and d must be in different subtrees may be not satisfied. Therefore we must subtract the expression 7.6 which is the number of times c and d are in the same subtree of v and u:

$$\sum_{k \neq i} \binom{|F_k \cap \bar{G}_j|}{2} + \sum_{l \neq j} \binom{|\bar{F}_i \cap G_l|}{2} - \sum_{k \neq i} \sum_{l \neq j} \binom{|F_k \cap G_l|}{2} \tag{7.6}$$

and the number of ways c and d can be found in different subtrees, given that a and b are found in $F_i \cap G_j$ is

$$\binom{|\bar{F}_i \cap \bar{G}_j|}{2} - \sum_{k \neq i} \binom{|F_k \cap \bar{G}_j|}{2} - \sum_{l \neq j} \binom{|\bar{F}_i \cap G_l|}{2} + \sum_{k \neq i} \sum_{l \neq j} \binom{|F_k \cap G_l|}{2}$$

In this way, for a pair of internal vertices $v \in V_1, u \in V_2$ the number of

butterfly topology quartets that are in both trees is:

$$shared(v,u) = \sum_i \sum_j \binom{|F_i \cap G_j|}{2} \left(\binom{|\bar{F}_i \cap \bar{G}_j|}{2} - \sum_{k \neq i} \binom{|F_k \cap \bar{G}_j|}{2} - \sum_{l \neq j} \binom{|\bar{F}_i \cap G_l|}{2} + \sum_{k \neq i} \sum_{l \neq j} \binom{|F_k \cap G_l|}{2} \right) \tag{7.7}$$

If the trees, T_1 and T_2, have a shared quartet $ab|cd$, then there are two internal nodes in each tree that claims this quartet. Due to that, the total number of shared quartets between the two trees is

$$shared(T_1, T_2) = \frac{1}{2} \sum_{v \in V_1 \setminus L} \sum_{u \in V_2 \setminus L} shared(v,u)$$

nonshared(T_1, T_2)
The number of butterfly quartets that are not shared by trees $T_1, T_2, nonshared(T_1, T_2)$ is the sum over each pair of internal nodes $v \in V_1$, $u \in V_2$ of $nonshared(v,u)$ - the number of butterfly quartets claimed by both internal nodes that are not common for the two trees. Such quartets have a property that a pair of leaves found in the same subtree of v will be found in different subtrees of u and vice versa, i.e. a nonshared quartet with leaves a, b, c, d can have $ab|cd$ topology for T_1 and $ad|bc$ topology for T_2 and $a \in F_i \cap G_j, b \in F_i \cap G_l, c \in F_k \cap G_n$ $d \in F_m \cap G_j$, $i \neq k \neq m, j \neq l \neq n$.

This is given by expression

$$nonshared(v,u) = \sum_i \sum_j \Bigg(|F_i \cap G_j||\bar{F}_i \cap G_j||F_i \cap \bar{G}_j||\bar{F}_i \cap \bar{G}_j| - \sum_{k \neq i} |F_i \cap G_j||F_i \cap \bar{G}_j||F_k \cap G_j||F_k \cap \bar{G}_j| - \sum_{l \neq j} |F_i \cap G_j||\bar{F}_i \cap G_j||F_i \cap G_l||\bar{F}_i \cap G_l| + \sum_{k \neq i} \sum_{l \neq j} |F_i \cap G_j||F_i \cap G_l||F_k \cap G_j||F_k \cap G_l| \Bigg) \tag{7.8}$$

where

$$\sum_i \sum_j |F_i \cap G_j||F_i \cap \bar{G}_j||\bar{F}_i \cap G_j||\bar{F}_i \cap \bar{G}_j| \tag{7.9}$$

is the number of all nonshared quartets related to a pair of nodes v and u, regarding that if two leaves of a quartet are in one subtree of v, they are in different subtrees of u and vice versa. However,we need to consider only those quartets that are claimed by v, u and this is fulfilled for a node v only when two leaves belong to one subtree of v (F_i), third leaf belongs to other subtree ($F_k, i \neq k$) and fourth belongs to another subtree ($F_m, i \neq k \neq m$). In 7.8 the first and second factor choose a and b from F_i, while the third and fourth choose c and d from $\bar{F}_i$. We must subtract from this expression the number of times when c and d are chosen from the same $F_k, k \neq i$ of v.

$$\sum_i \sum_j \sum_{k \neq i} |F_i \cap G_j||F_i \cap \bar{G}_j||F_k \cap G_j||F_k \cap \bar{G}_j| \tag{7.10}$$

and similarly for cases where b and c are chosen from the same $G_l, l \neq j$ of u, we subtract

$$\sum_i \sum_j \sum_{l \neq j} |F_i \cap G_j||\bar{F}_i \cap G_j||F_i \cap G_l||\bar{F}_i \cap G_l| \tag{7.11}$$

Finally, both 7.10 and 7.11 substract the same quartets for which in 7.9 both c, d are chosen from the same $F_k, k \neq i$ of v and b, c are chosen from the same $G_l, l \neq j$ of u. We we must therefore add to 7.9 an expression

$$\sum_i \sum_j \sum_{k \neq i} \sum_{l \neq j} |F_i \cap G_j||F_i \cap G_l||F_k \cap G_j||F_k \cap G_l|$$

Assuming that the trees have a nonshared quartet with topology form $ab|cd$ in T_1, and $ad|bc$ in T_2, there are two internal nodes in each tree claiming the quartet: $ab|cd$ in T_1 and $ad|bc$ in T_2. Therefore the total number of nonshared by trees T_1, T_2 quartets is

$$nonshared(T_1, T_2) = \frac{1}{4} \sum_{v \in T_1 \setminus L} \sum_{u \in T_2 \setminus L} nonshared(v, u)$$

Performance

The [15] gives more detailed algorithm description, where using dynamic dynamic programming, the authors drive equations 7.7 and 7.8 to forms that

allow for the $O|L| + |V_1 \setminus L||V_2 \setminus L|min\{id_1, id_2\})$ time. Briefly speaking, in order to get $shared(T_1, T_2)$ and $nonshared(T_1, T_2)$, the algorithm for every subtree in T_1 visits every subtree in T_2, performing operations in time $O(min\{id_1, id_2\})$.

As [15] claims, the worst theoretical running time of the algorithm for calculating the quartet distance presented above is $O(|L|^3)$ (for a tree with an internal node of degree $\frac{|L|}{2}$, connected to $\frac{|L|}{2}$ internal nodes of degree three, each connected to two leaves). However, basing on the experiments documented in the article, the authors claim that for trees used in practice the algorithm runs in time $O(|L|^2)$.

7.4 Triplets distance

The triplets distance is a measure of distance between two rooted bifurcating trees. It counts the number of subtrees of three taxa that are different in the trees. The distance is called in [18] a close cousin of the quartet metric for unrooted trees.

TAGS
triplets, rooted tree

INPUT DATA
Two rooted phylogenetic trees $T_1, T_2 \in T_L$ with the same set of leaves L.

DEFINITION
The triplets distance $d(T_1, T_2)$ between two phylogenetic trees T_1 and T_2 is defined as follows:
For each triple $\{i, j, k\}$ of distinct leaves (taxa) in the leaves set, we consider the subtrees of T_1 and T_2 that relates these three taxa alone. There are exactly four possibilities for this subtree: the star topology and the three that differ in which of i, j, k taxon is the most distant leaf relative to the other two. We define an indicator function as follows:

$$I_{i,j,k} = \begin{cases} 1 & \text{taxa } i,\ j,\ k \text{ have different subtrees for the two trees} \\ 0 & \text{taxa } i,\ j,\ k \text{ have the same subtrees for the two trees} \end{cases}$$

We define the triplets distance as follows:

$$d = \sum_{i,j,k} I_{i,j,k}$$

where the summation is over $\binom{|L|}{3}$ possible triplets i, j, k of $|L|$ distinct taxa.

Table 7.11 shows this definition in the nomenclature from section 7.1.

function	description
$f(T)$	returns a set of all T's triplets
h	returns 1 if a triplet of one tree has different topology in the other tree, 0 otherwise
g_h	$g_{h,1}(f(T_1), f(T_2)) = \sum_{t \in f(T_1)} h(t, f(T_2)) + \sum_{t \in f(T_2)} h(t, f(T_1))$

Table 7.11: Definition of the triplets distance in the nomenclature from section 7.1.

THE $O(n^2)$ ALGORITHM

Critchlow et al. in [18] introduce an efficient and easy to implement $O(n^2)$ algorithm ($n = |L|$) for binary trees case that bases on generational matrix tree representation and a particular observation. The disadvantage of the algorithm is the constraint for input trees to be binary.

Observation: For any triplet $\{i, j, k\}$ of taxa, i is the most distant leaf for the triplet if and only if the most recent common ancestor in the tree for both $\{i, j\}$ and $\{i, k\}$ is the same. E.i. in the figure 7.6, '7' is the most distant leaf for the {2,4,7} triplet: the most recent common ancestor between leaves {2,4} is internal node a whereas for {2,7} and for {4,7} it is the same internal node b.

Generational matrix: a matrix that for each pair of leaves in a tree, stores the level of the most recent common ancestor (the level of the node at which the two taxa split). By the level of a node we mean the number of branches on the path from the root to the node. The tables 7.12 and 7.13 are examples of generational matrices for respectively trees T_1 and T_2 from the figure 7.7

By *generational pattern* for leaves i, j we will denote an ordered pair which first element is a value of T_1's generational matrix at (i, j) and which second element is a value of T_2's generational matrix at (i, j).

For trees T_1, T_2 and a triplet $\{i, j, k\}$, the i leaf is the most distant for both triplets' topologies if and only if the generational pattern for i, j equals to the generational pattern for i, k. This means the triplet has the same topology for both trees. The algorithm inspects generational patterns for each leaf (which leads to scanning either rows or columns of both generational matrices),

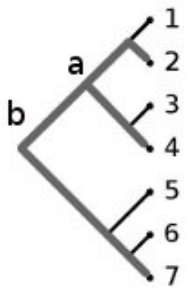


Figure 7.6: A rooted binary tree.

computing the occurrences of same generational patterns for that leaf. If there are two occurrences of the same pattern - the same triplet topology is found; if there are 3 occurrences - there are $\binom{3}{2}$ same triplets found; if there are k occurrences - there are $\binom{k}{2}$ same triplets found.

As in a tree there are $\binom{n}{3}$ triplets, if we reduce that number with the number of all same topology triplets, we get the required triplets distance value.

The procedure above takes $O(n^2)$ time. This can be achieved e.i. by for each leaf, filling a matrix where the stored values are the numbers of times that generational patterns are associated with a leaf; for a sample see table 7.14. The size of matrix is a square of $maxH$ - the maximum from T_1, T_2 heights (which can maximally be n). The value in row a column b responds to the number of times the generational pattern (a, b) appears for a currently considered leaf.

To get quick access to only generational patterns that repeated at least twice, we can take advantage of e.i. a stack: if a generational pattern appears twice, the reference to the matrix' field that keeps the number of occurrences is added to the stack. When the leaf's examination finishes, the algorithm goes through the stack elements and subtracts binomial coefficients of the values in matrix' fields pointed by the stack elements. For the sample corresponding to the table 7.14 see the table 7.15.

As each leaf can be assigned to maximally $\frac{1}{2}maxH$ generational patterns and as there are n leaves; also as construction of generational matrix can take $O(n^2)$ time – the total complexity of the algorithm is $O(n^2)$.

SAMPLE
Consider two trees T_1, T_2 on figure 7.7. Their generational matrices, $M(T_1), M(T_2)$

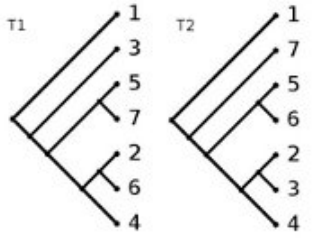


Figure 7.7: Two binary rooted trees T_1 and T_2 with the same taxa set.

are respectively the 7.12 and 7.13 tables. Note that the generational pattern (1,1) is repeated six times for the first leaf (scanning the row with id 1 for both matrices), the pattern (2,3) occurs two times for the leaf 3, (3,3) appears two times for the leaf 5, (3,2) occurs three times for the leaf 7. The sample matrix with the number of times that an arbitrary chosen leaf 7 is associated with generational patterns is presented as table 7.14 and its stack - table 7.15 - holds the reference to only the (3,2) field.

For the two considered trees, the triplets distance is $d = \binom{n}{3} - \binom{6}{2} - \binom{2}{2} - \binom{2}{2} - \binom{3}{2} = 35 - 15 - 1 - 1 - 3 = 15$.

id	*1*	*2*	*3*	*4*	*5*	*6*	*7*
1	-	0	0	0	0	0	0
2	0	-	1	3	2	4	2
3	0	1	-	1	1	1	1
4	0	3	1	-	2	3	2
5	0	2	1	2	-	2	3
6	0	4	1	3	2	-	2
7	0	2	1	2	3	2	-

Table 7.12: A generational matrix for the tree T_1. The first row and the first column are the leaves id.

id	*1*	*2*	*3*	*4*	*5*	*6*	*7*
1	-	0	0	0	0	0	0
2	0	-	4	3	2	2	1
3	0	4	-	3	2	2	1
4	0	3	3	-	2	2	2
5	0	2	2	2	-	3	2
6	0	2	2	2	3	-	1
7	0	1	1	1	1	1	-

Table 7.13: A generational matrix for the tree T_2. The first row and the first column are the leaves id.

pattern	*0*	*1*	*2*	*3*	*4*
0	1	0	0	0	0
1	0	1	**3**	1	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Table 7.14: The matrix that stores the number of times generational patterns for the leaf 7 are repeated. The bolded 3 value is the only one that will be considered when computing the distance; this is the value to which points the element in the stack (table 7.15).

row:1 col:2

Table 7.15: The stack for the 7.14 matrix - the matrix for leaf '7'. It points on all the elements in the matrix that have value at least 2 - which is the element in the row 1 column 2 with value '3'.

7.5 Nodal distance

The nodal distance, also called the path difference, compares the phylogenetic trees considering the branch distances between nodes. In [4] John Bluis and Dong-Guk Shin describe the metric as the analysis of changes that occur in migration of spices between two trees. They claim its advantage is to be insensitive to some trees differences that seem to be not meaningful in phylogenetic context, but which the most popular partitioning algorithms

consider significant (for details, see [4]).

TAGS
unrooted tree, path

INPUT DATA
Two unrooted phylogenetic trees $T_1, T_2 \in T_L$ with the same set of leaves L.

DEFINITION
Having two trees T_1, T_2, the nodal distance is a is a p-norm on the difference for two trees in a particular pair of leaves' branch distances,
where by a *branch distance* $l(i, j)$ we denote the number of branches on the path between the pair of leaves i, j.

Table 7.16 shows this definition in the nomenclature from section 7.1.

function	description
$f(T)$	returns a $\binom{\lvert L\rvert}{2}$-sized vector of branch distances between every possible pair of leaves
h	returns the difference between branch distances built on the same pair of leaves
g_h	$g_{h,p}(f(T_1), f(T_2)) = (\sum_{i,j \in V(T_1), i \neq j} \lvert l_1(i,j) - l_2(i,j)\rvert^p)^{\frac{1}{p}}$

Table 7.16: Definition of the nodal distance in the nomenclature from section 7.1.

The popular choices of p parameter for p-norm are $p = 1$ and $p = 2$. They are commonly denoted as **manhattan** or **taxicab metric** ($p = 1$) and **pythagorean metric** ($p = 2$).

THE $O(n^2)$ ALGORITHM
We can compute the nodal distance in $O(n^2)$ time:

1. For each of the two compared trees, for each of $\binom{n}{2}$ pairs leaves - compute and store the branch distance. This can be done in $O(n^2)$ time - e.i. by calling for each of n leaves as a root the $O(n)$ DFS algorithm.

2. Compute a p-norm on the $\binom{n}{2}$-elements vector of differences between each pair of leaves' branch difference in trees T_1 and T_2.

WEIGHTED NODAL
If the input trees have weighted edges, we can compute the nodal distance

with regard to those weights. In this case the branch distance between a pair of leaves is the sum of weights on branches that build a path between the pair of leaves.

Notice that if every edge in both trees has weight 1, then the weighted nodal distance between the trees is equal to the not considering edges weights nodal distance.

7.6 Correctness of definition

In this section we present the proofs that the methods described in this chapter are metrics in a space of phylogenetic trees.

Lets recall the three conditions that a function $d : X \times X \to R$ must fulfil in order to be a *metric* on space (T_L, d). For $T_A, T_B, T_C \in T_L$:

1. Positiveness: $d(T_A, T_B) > 0$ if $T_A \neq T_B$, and $d(T_A, T_A) = 0$.
2. Symmetry: $d(T_A, T_B) = d(T_B, T_A)$.
3. Triangle inequality: $d(T_A, T_B) \leq d(T_A, T_C) + d(T_B, T_C)$.

As all the metrics in this chapter have form of p-norms on the space of vectors with distances between two trees' description elements sets, due to the norm properties (see 4.2) all the metrics fulfil the *triangle inequality* condition.

As for the *symmetry* condition: for each of the Robinson-Foulds, quartets and triplets metrics, a method of comparing description elements sets is to check if a description element exists in other tree's elements set: the distance is 0 if *yes* and 1 if *no*. This is not dependant from the sequence of parameters and $d(T_A, T_B) = d(T_B, T_A)$. Similarly for the nodal metric, the difference between branch distances is not dependant from the sequence of parameters.

As a tree can be reconstructed from the set of all this is true for description elements and bipartitions, cluster, quartets and triplets [12] as well as for branch distances[1], $d(T_1, T_2) = 0$ only if $T_1 = T_2$ and for $T_1 \neq T_2$ from the norm properties, $d(T_1, T_2) > 0$ and therefore all the distances fulfil the *positiveness* condition.

[1]Having branch distances between each pair of leaves, we can reconstruct a tree e.i. with Neighbour Joining method, see section 5.4

Chapter 8

Minimum Weight Perfect Matching Distance

The Gdansk University of Technology has recently (in 2008) presented a *Minimum weight Perfect Matching Distance* (MWPMD) - the new technique of computing the distance between phylogenetic trees [5].

In general, MWPMD is the family of metrics; its assumptions are presented in the first section of this chapter. In the further sections we present certain solutions for the family.

8.1 General MWPMD assumptions

The proceedings for the family of MWPM metrics base on finding a one-to-one mapping between elements from two sets where each set in an explicit way describes the phylogenetic tree. The mapping minimizes certain criterion and provides weights to find a minimum weight perfect matching cost between the trees. This cost is the Minimum Weight Perfect Matching Distance.

INPUT DATA

- Two phylogenetic trees $T_1, T_2 \in T_L$ with the same set of leaves L. The specification of trees topology e.i. rooted/unrooted depends on a choice of function f.
- A (polynomial time computable) function $\boldsymbol{f}$ that assigns a tree to a finite set of description elements.

- A (polynomial time computable) function $\boldsymbol{h}$ - a metric that measures the distance between description elements of different trees. If f allows for returning different number of description elements for T_1, T_2 then there is also a need to define a rule which determines the distance between a description element and an element that equalizes the difference between the sizes of description elements sets, called a *dummy element*.

DEFINITION
Here we describe the general steps for the family of metrics.

The general definition is as follows:

$$d_{f,h}(T_1, T_2) = w(M_h(f(T_1), f(T_2))) \tag{8.1}$$

where:

- $T_1, T_2 \in T_L$, where T_L is a family of phylogenetic trees on set of species (leaves) L.
- function $f : T_L \to 2^{D \setminus \{O\}}$ assigns a finite set of description elements (elements from D) to a tree.
  A description element is a piece of information about topology of a tree, for example, it can be a branch, split or cluster in some phylogenetic tree description. We denote the set of all description elements as $D \setminus \{O\}$.
- O is a *dummy element* $O \in D$: function f returns a set of elements. If for a pair of trees T_1, T_2, $|f(T_1)| \neq |f(T_2)|$, then the smaller set is extended by dummy elements until the sets are of equal size.
- function $h : D \times D \to R_+ \cup \{0\}$ is a metric on elements from D.
- M_h is a minimum weight perfect matching on a complete bipartite graph $G = (V, E)$. Weights of edges in G depend on function h.
- $w(M_h)$ is the weight of the matching M_h.

This definition is equivalent the definition from section 7.1, see table 8.1.

According to the equation 8.1, the steps in MWPM distance algorithm are following:

function	MWPM equivalent
$f(T)$	$f(T)$
h	h
$g_h(f(T_1), f(T_2))$	$w(M_h(f(T_1), f(T_2)))$

Table 8.1: Definition of the MWPM distance in the nomenclature from section 7.1.

1. With use of function f acquire two independent description elements sets $V_{T_1}, V_{T_2} \in D$ on respectively trees T_1, T_2. They constitute the two graph G partitions.
   If V_{T_1}, V_{T_2} are of different size - extend the smaller partition of G by the missing number of dummy vertices which correspond to a dummy element $O \in D$

2. Connect every description element $v_{T_1} \in V_{T_1}$ with every description element $v_{T_2} \in V_{T_2}$ and give each connection a weight according to h. At this point we have a complete bipartite graph G with weights (see figure 8.1).

3. Find the minimum weight perfect matching on G and return its weight as the resultant distance between the trees.

TIME COMPLEXITY
As the following are performed as separate steps:

- $f(T)$ - assigning a finite set of description elements to T.
- h - computing the distance between each pair of vertices from different partitions
- M - determining perfect matching minimum weight for a complete bipartite graph

The time complexity of the MWPMD algorithm depends on a definition of $f(T)$ and h functions and is lower bounded by the complexity of the minimum weight perfect matching algorithm in bipartite graphs. For a bipartite graph $G(V, E)$, the minimum weight perfect matching algorithm can be performed in $O(|E|\sqrt{|V|}\log(|V|max_{e\in E}w(e))$ [7], where $w(e)$ is the weigth of $e \in E$. In this paper though, we investigate a Hungarian algorithm that works in $O(|V|^3)$ time (see section 8.5 page 100).

CORRECTNESS OF DEFINITION
Here we briefly characterize the metrics:

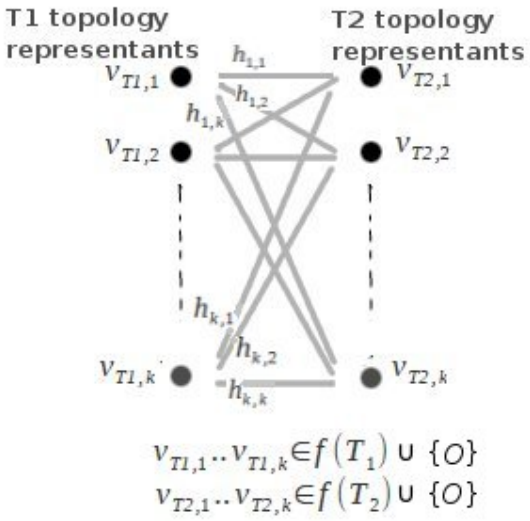


Figure 8.1: A complete bipartite graph G. Each of $2k$ ($k = |D|$) vertices corresponds to an element from D – a description element.

1. all the description elements of a tree respond to a bipartition in a complete bipartite graph G: all the elements from one partition are adjacent to every element from the other tree. The number of elements in each bipartition is the same. If elements sets have different number of elements, we add dummy elements O.
2. every edge in G is weighted with a value that is returned by a metric h on two different tree's description elements (so an edge's weight fulfils metric conditions).
3. the two trees distance function d is a weight of a minimum weight perfect matching on a bipartite graph G.
4. formally (3) is $d(T_i, T_j) = \sum_{l=1}^{N} w_{i,j}(l)$ where $w_{i,j}(l)$ denotes the weight of an edge number l in a minimum weight perfect matching in a bipartite graph G, which partitions correspond respectively to T_i's, T_j's description elements set.
5. $h(O, O) = 0$ as the h function that computes the weights is a metric

Due to (2) and (3), the *positiveness* and *symmetry* conditions are fulfilled.

To prove the *triangle inequality* condition we firstly equalize the number of description elements for all the trees T_1, T_2, T_3, which does not influence

the total distance result as (5). From (4) we have $d(T_1, T_2) + d(T_2, T_3) = \sum_{l=1}^{N} w_{1,2}(l) + \sum_{l=1}^{N} w_{2,3}(l)$ and $d(T_1, T_3) = \sum_{l=1}^{N} w_{1,3}(l)$. We have to prove that $\sum_{l=1}^{N} w_{1,2}(l) + \sum_{l=1}^{N} w_{2,3}(l) \geq \sum_{l=1}^{N} w_{1,3}(l)$ (see fig 8.2), which is true recalling that h is a metric and $w_{1,2}(l) + w_{2,3}(l) \geq w_{1,3}(l)$.

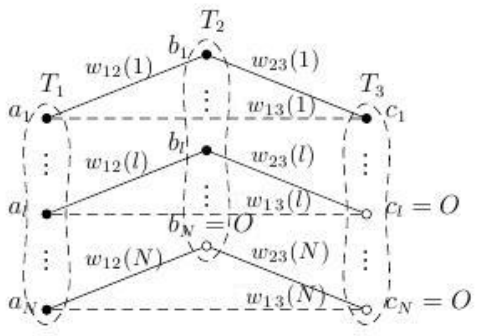


Figure 8.2: Illustration for the triangle inequality proof. The perfect matching graph between description elements $a_l \in f(T_1)$ and $b_l \in f(T_2)$ joined with perfect matching of $c_l \in f(T_3)$ and $b_l \in f(T_2)$. Each of T_1, T_2, T_3 trees has the same number of description. The labels of the description elements are sorted such that if a_l is matched to b_l and c_k is matched to b_l then $k = l$. The label l of two description elements corresponds also to the label that is given to an edge which joins the two elements. (Source: modified [5])

Special Cases of the Minimum Weight Perfect Matching Distance

In the next tree sections of this chapter we present certain solutions for the family of MWPMD. This implies providing definitions for the f and h functions from the 8.1 expression (page 86).

The two prior methods base on partitioning the tree into two sets and qualifying the dichotomy of leaves. The last method compares the internal nodes in context of being the most recent common ancestors for leaves' pairs. For more information, consult also [5], [6], [7].

8.2 Split MWPM distance

INPUT DATA
Two unrooted phylogenetic trees $T_1, T_2 \in T_L$ with the same set of leaves L.

FUNCTION *f*
In the f definition $f : T_L \to 2^{D \setminus \{0\}}$, an element from D stands for a non-trivial split also called a bipartition.

According to a split definition (section 7.2), the number of non-trivial splits of leaves L in a tree T_L is equal to the number of internal edges.

FUNCTION *h*
Let $A_1|B_1$ and $A_2|B_2$ be single splits of respectively T_1, T_2. We define a metric h on splits as:

$$h(A_1|B_1, A_2|B_2) = 0.5min(|A_1 \oplus A_2| + |B_1 \oplus B_2|, |A_1 \oplus B_2| + |B_1 \oplus A_2|) \quad (8.2)$$

Notice that

- $X \oplus Y = (L \setminus X) \oplus (L \setminus Y)$
- for a bipartition $A|B : A = L \setminus B$

therefore for equation 8.2 $A_1 \oplus A_2 = B_1 \oplus B_2$. This implies

$$h(A_1|B_1, A_2|B_2) = min(|A_1 \oplus A_2|, |A_1 \oplus B_2|) \quad (8.3)$$

Let $|L|$ be the number of leaves in tree T_L. As $|A_1 \oplus B_2| = |A_1 \oplus (L \setminus A_2)| = |L| - |(A_1 \oplus A_2)|$ We can drive equation 8.2 to

$$h(A_1|B_1, A_2|B_2) = min(|A_1| + |A_2| - 2|A_1 \cap A_2|, |L| - (|A_1| + |A_2| - 2|A_1 \cap A_2|) \quad (8.4)$$

h is a metric in space of splits as equation 8.2 is independent from the sequence of arguments; equal to 0 only if $A_1|B_1 = A_2|B_2$, otherwise greater than 0; as for the *triangle inequality* - we must prove all the 8 options. We list them in table 8.2, providing equation referenced in the table. We reduce the number of equations, noticing that the equations for the last two options are analogous to the two other. We also take advantage of the fact, that some equation may be driven to others.

equation	$min(\lvert A_1 \oplus A_2\rvert, \lvert A_1 \oplus B_2\rvert)$	$min(\lvert A_2 \oplus A_3\rvert, \lvert A_2 \oplus B_3\rvert)$	$min(\lvert A_1 \oplus A_3\rvert, \lvert A_1 \oplus B_3\rvert)$
8.5	$\lvert A_1 \oplus A_2\rvert$	$\lvert A_2 \oplus A_3\rvert$	$\lvert A_1 \oplus A_3\rvert$
8.8, driven to 8.5	$\lvert A_1 \oplus A_2\rvert$	$\lvert A_2 \oplus A_3\rvert$	$\lvert A_1 \oplus B_3\rvert$
8.6	$\lvert A_1 \oplus B_2\rvert$	$\lvert A_2 \oplus B_3\rvert$	$\lvert A_1 \oplus A_3\rvert$
8.9, driven to 8.6	$\lvert A_1 \oplus B_2\rvert$	$\lvert A_2 \oplus B_3\rvert$	$\lvert A_1 \oplus B_3\rvert$
8.7	$\lvert A_1 \oplus A_2\rvert$	$\lvert A_2 \oplus B_3\rvert$	$\lvert A_1 \oplus B_3\rvert$
8.10, driven to 8.7	$\lvert A_1 \oplus A_2\rvert$	$\lvert A_2 \oplus B_3\rvert$	$\lvert A_1 \oplus A_3\rvert$
analogous to 8.7	$\lvert A_1 \oplus B_2\rvert$	$\lvert A_2 \oplus A_3\rvert$	$\lvert A_1 \oplus B_3\rvert$
analogous to 8.10	$\lvert A_1 \oplus B_2\rvert$	$\lvert A_2 \oplus A_3\rvert$	$\lvert A_1 \oplus A_3\rvert$

Table 8.2: Alternative options for expression 8.3 assigned to equations that prove the triangle inequality for 8.3.

$$
\begin{aligned}
&|A_1 \oplus A_2| + |A_2 \oplus A_3| \geq |A_1 \oplus A_3| \\
\iff &|A_1|+|A_2|-2|A_1 \cap A_2|+|A_2|+|A_3|-2|A_2 \cap A_3| \geq |A_1|+|A_3|-2|A_1 \cap A_3| \\
\iff &|A_2| \geq |A_1 \cap A_2| + |A_2 \cap A_3| - |A_1 \cap A_3| \\
\iff &|A_2 \setminus A_1 \setminus A_3| \geq 0
\end{aligned}
\tag{8.5}
$$

$$
\begin{aligned}
&|A_1 \oplus B_2| + |A_2 \oplus B_3| \geq |A_1 \oplus A_3| \\
\iff &|L| - (|A_1| + |A_2| - 2|A_1 \cap A_2|) + |L| - (|A_2| + |A_3| - 2|A_2 \cap A_3|) \\
&\quad \geq |A_1| + |A_3| - 2|A_1 \cap A_3| \\
\iff &|L| + |A_1 \cap A_2| + |A_2 \cap A_3| + |A_1 \cap A_3| \geq |A_1| + |A_2| + |A_3| \\
\iff &|L| + |A_1 \cap A_2| + |A_2 \cap A_3| + |A_1 \cap A_3| \\
&\quad \geq |L| + |A_1 \cap A_2| + |A_2 \cap A_3| + |A_1 \cap A_3| - |A_1 \cap A_2 \cap A_3| \\
\iff &|A_1 \cap A_2 \cap A_3| \geq 0
\end{aligned}
\tag{8.6}
$$

$$
\begin{aligned}
&|A_1 \oplus A_2| + |A_2 \oplus B_3| \geq |A_1 \oplus B_3| \\
\iff &|A_1| + |A_2| - 2|A_1 \cap A_2| + |L| - (|A_2| + |A_3| - 2|A_2 \cap A_3|) \\
&\quad \geq |L| - (|A_1| + |A_3| - 2|A_1 \cap A_3|) \\
\iff &|A_1| \geq |A_1 \cap A_3| + |A_1 \cap A_2| - |A_2 \cap A_3| \\
\iff &|A_1 \setminus A_2 \setminus A_3| \geq 0
\end{aligned}
\tag{8.7}
$$

$$
\begin{aligned}
&|A_1 \oplus A_2| + |A_2 \oplus A_3| \geq |A_1 \oplus B_3| \\
\impliedby &|A_1 \oplus A_2| + |A_2 \oplus A_3| \geq |A_1 \oplus A_3| \text{ (see equation 8.5).}
\end{aligned}
\tag{8.8}
$$

$$|A_1 \oplus B_2| + |A_2 \oplus B_3| \geq |A_1 \oplus B_3| \\ \Longleftarrow |A_1 \oplus B_2| + |A_2 \oplus B_3| \geq |A_1 \oplus A_3| \text{ (see equation 8.6).} \tag{8.9}$$

$$|A_1 \oplus A_2| + |A_2 \oplus B_3| \geq |A_1 \oplus A_3| \\ \Longleftarrow |A_1 \oplus A_2| + |A_2 \oplus B_3| \geq |A_1 \oplus B_3| \text{ (see equation 8.7).} \tag{8.10}$$

FUNCTION h FOR DUMMY ELEMENTS

The special case of h function that defines any distance to a dummy vertex must be defined in a way it fulfils the condition required for a metric in space of splits. Bogdanowicz in [7] defines two variants that fulfils requirements.

$$h_1(A|B,O) = h_1(O,A|B) = \lceil \lfloor \frac{|L|}{2} \rfloor /2 \rceil, h_1(O,O) = 0$$

$$h_2(A|B,O) = h_2(O,A|B) = min(|A|,|B|), h_1(O,O) = 0$$

h_1 and h_2 fulfil the *positiveness* metric condition, as the result is equal to 0 only if called on two dummy elements and non-negative otherwise. The *symmetry* condition is by definition.

As for the *triangle inequality* for h_1, regarding equation 8.4, the maximal value for h for splits is $h(A_1|B_1, A_2|B_2) \leq \lfloor \frac{|L|}{2} \rfloor$. Among the seven possible triangle inequalities containing dummy elements, the only non-trivial case is $h_1(A_1|B_1,O) + h_1(O,A_2|B_2) = 2\lceil \lfloor \frac{|L|}{2} \rfloor /2 \rceil \geq \lfloor \frac{|L|}{2} \rfloor \geq h(A_1|B_1,A_2|B_2)$.

As for the *triangle inequality* for h_2,
$h_2(A_1|B_1,O) + h_2(O,A_2|B_2) = min(|A_1|,|B_1|) + min(|A_2|,|B_2|)$
which is equal to one of $|A_1| + |A_2|, |A_1| + |B_2|, |B_1| + |A_2|, |B_1| + |B_2|$.
This is always greater than $h(A_1|B_1,A_2|B_2)$ as

$$h(A_1|B_1,A_2|B_2) = \begin{cases} min(|A_1 \oplus A_2|, |A_1 \oplus B_2|) \leq \begin{cases} |A_1| + |A_2| \\ |A_1| + |B_2| \end{cases} \\ \\ min(|B_1 \oplus A_2|, |B_1 \oplus B_2|) \leq \begin{cases} |B_1| + |A_2| \\ |B_1| + |B_2| \end{cases} \end{cases}$$

TIME COMPLEXITY

The in Robinson-Foulds for unrooted trees algorithm for f presented in section 7.2 has the $O(n^2)$ time complexity ($n = |L|$). The h algorithm proposition is very similar to the second step of 7.2 section, with the difference that there is count a number of leaves that are different for description elements; the time complexity for this naive algorithm is $O(n^3)$. The complexity of Hungarian algorithm depends on the number of description elements which is equal to the number of internal edges and therefore the hungarian algorithm is $O(n^3)$. The total complexity is $O(n^3)$.

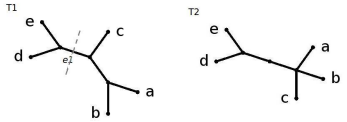


Figure 8.3: Unrooted trees T_1 and T_2. The dashed line symbolises the split on e_1 edge that divides T_1 into partitions: $\{a, b, c\}$ and $\{d, e\}$.

element of $D_1 \setminus D_2$	{a,b,c}|{d,e}	O_1
{a,b,c}|{d,e}	0	1 (for h_1) 2 (for h_2)
{a,b}|{c,d,e}	1	1 (for h_1) 2 (for h_2)

Table 8.3: Distances between description elements from D_1 and D_2.

EXAMPLE
Figure 8.3 presents two unrooted trees T_1, T_2 on a set of taxons $L = \{a, b, c, d, e\}$.

There are two non-trivial bipartitions in T_1: $D_1 = \{\{a, b, c\}|\{d, e\}, \{a, b\}|\{c, d, e\}\}$; there is one non-trivial bipartition in T_2: $D_2 = \{\{a, b, c\}|\{d, e\}\}$. As $|D_2| = |D_1| - 1$, we add one dummy element to D_2, so that $D_2 = \{\{a, b, c\}|\{d, e\}, O_1\}$.

Table 8.3 shows calculations of distances between description elements from D_1 and D_2. The split minimum weight perfect matching distance is in this case 1 if using h_1 for computing distances to dummy element and 2 if using h_2 (see fig 8.4).

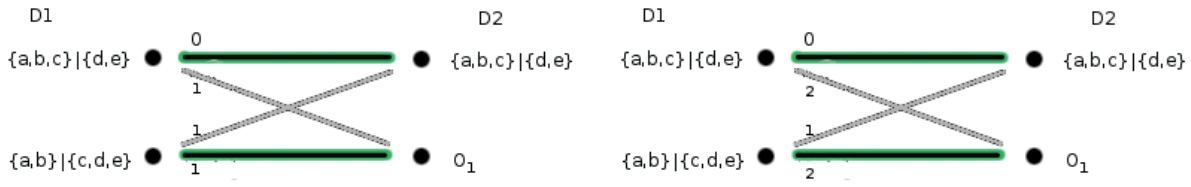


Figure 8.4: Two possibilities of bipartite graph G - according to the h function for dummy elements choice: the graph on the left pictures the h_1 variant, the graph on the right - the h_2's one. Vertices are the description elements, weighted edges are the distances between D_1 and D_2 elements.

8.3 Clusters MWPM distance

INPUT DATA
Two rooted phylogenetic trees $T_1, T_2 \in T_L$ with the same set of leaves L.

FUNCTION *f*
In the f definition $f : T_L \to 2^{D\setminus\{0\}}$, an element from D stands for a cluster.

According to a cluster definition (section 7.2), the number of clusters in a tree T_L is equal to the number of internal nodes.

FUNCTION *h*
Let A_1 and A_2 be single clusters in respectively T_1, T_2. We define a metric on clusters:

$$h(A_1, A_2) = |A_1 \oplus A_2| = |A_1| + |A_2| - 2|A_1 \cap A_2| \tag{8.11}$$

h is a metric in space of clusters as equation 8.11 is independent from the sequence of arguments; equal to 0 only if $A_1 = A_2$, otherwise greater than 0; for three clusters A_1, A_2, A_3:
$h(A_1, A_2) + h(A_2, A_3) \geq h(A_1, A_3)$
$\iff |A_1|+|A_2|-2|A_1 \cap A_2|+|A_2|+|A_3|-2|A_2 \cap A_3| \geq |A_1|+|A_3|-2|A_1 \cap A_3|$
$\iff |A_2| \geq |A_1 \cap A_2| + |A_2 \cap A_3| - |A_1 \cap A_3|$
which is true.

FUNCTION *h* FOR DUMMY ELEMENTS
Just as in section 8.2, we propose when a dummy element is an argument two variants of h [7] (A - a cluster-description element, O - dummy element, $|L|$ - number of leaves):

$$h_1(A, O) = h_1(O, A) = \lceil \frac{|L|}{2} \rceil, h_1(O, O) = 0$$

$$h_2(A, O) = h_2(O, A) = |A|, h_2(O, O) = 0$$

h_1 and h_2 fulfil the *positiveness* metric condition, as the result is equal to 0 only if called on two dummy elements and non-negative otherwise. The *symmetry* condition is by definition.

As for the *triangle inequality*, regarding equation 8.11, the maximal value for h for clusters A_1 from T_1, A_2 from T_2 is $h(A_1, A_2) \leq |L|$. Among the seven possible triangle inequalities containing dummy elements, the only non-trivial case is regarding h_1

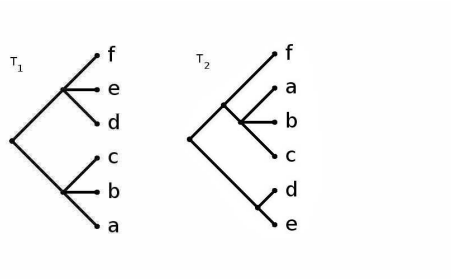


Figure 8.5: Rooted trees T_1 and T_2.

element of $D_1 \setminus D_2$	{a,b,c}	{a,b,c,f}	{d,e}
{a,b,c}	0	1	5
{d,e,f}	6	5	1
O_1	2 (for h_1) 3 (for h_2)	2 (for h_1) 4 (for h_2)	2 (for h_1) 2 (for h_2)

Table 8.4: Distances between description elements from D_1 and D_2.

$h_1(A_1, O) + h_1(O, A_2) = 2\lceil \frac{|L|}{2} \rceil \geq |L| \geq h(A_1, A_2)$.
and regarding h_2
$h_2(A_1, O) + h_2(O, A_2) = |A_1| + |A_2| \geq |A_1| + |A_2| - 2|A_1 \cap A_2| = h(A_1, A_2)$.

TIME COMPLEXITY
The f algorithm proposition is presented in section 7.2 (Robinson-Foulds algorithm) in "The general algorithm and its complexity" subsection's first step description, rooted trees case and it has the $O(n^2)$ time complexity. The h algorithm proposition is very similar to the second step of 7.2 section, with the difference that there is count a number of leaves that are different for description elements; the time complexity is $O(n^3)$.

EXAMPLE
Figure 8.5 presents two rooted trees T_1, T_2 on a set of taxons $L = \{a, b, c, d, e, f\}$.

There are two clusters in $T_1 : D_1 = \{\{a, b, c\}, \{d, e, f\}\}$ and three clusters in $T_2 : D_2 = \{\{a, b, c\}, \{a, b, c, f\}, \{d, e\}\}$. We add a dummy vertex O_1 to D_1 make the cluster sets equal in size: $D_1 = \{\{a, b, c\}, \{d, e, f\}, O_1\}$. .

Table 8.4 shows calculations of distances between description elements from D_1 and D_2. The cluster minimum weight perfect matching distance is in this case 3 if using h_1 for computing distances to dummy element and 5 if using h_2.

8.4 Pairs MWPM distance

INPUT DATA
Two rooted binary phylogenetic trees $T_1, T_2 \in T_L$ with the same set of leaves L.

FUNCTION *f*
In the f definition $f : T_L \to 2^{D \setminus \{0\}}$, an element from D stands for a set unordered pairs of leaves. It is obtained in the following way:

Each of the possible $\binom{n}{2}$ pairs of leaves from the leaves set $L : |L| = n$ is assigned to an internal node which is the pair's most recent common ancestor. The set of all the pairs of an internal node form a description element.

Some properties:

- for any rooted binary input tree $T \in T_L$ there are always $n - 1$ description elements returned by $f(T)$ and there is no need for dummy elements.
- each internal node is a most recent common ancestor for at least 1 pair of leaves.

FUNCTION *h*
The distance between two description elements is the number of pairs that occur in exactly one of the two description elements' set of unordered pairs. Formally,

$$h(A_1, A_2) = |(A_1 \cup A_2) \setminus (A_1 \cap A_2|) = |A_1 \oplus A_2|$$

where A_k is a description element - a set of unordered pairs for a particular internal node of tree T_k

For the proof of the metric properties of h, see section 8.3.

FUNCTION *h* FOR DUMMY ELEMENTS
As both compared trees are bifurcating, the number of description elements for both trees is the same.

THE f AND h ALGORITHM AND ITS COMPLEXITY
We propose the algorithm that selects description elements and computes the distances between different trees' elements in $O(n^2)$ time.[1] It does not

[1]However, the whole pairs MWPM complexity depends also on finding the minimum weight perfect matching, therefore as currently there is no algorithm that performs so in $O(n^2)$ time, the total complexity is the minimum weight perfect matching's one.

distinguish separate steps for f and h, but within 2 steps results in a matrix with distances between description elements of different trees:

1. Building leaves' most recent common ancestor matrix for each tree.

   The matrix labels are the ids of leaves, the $matrix(i, j)$ value is the id of an internal node that is the most recent common ancestor for leaves i, j (see tables 8.5 and 8.6).

2. Building matrix with the number of occurrence for each *generational pattern*.

   Having two trees T_1, T_2 and a pair of leaves $\{i, j\}$, we call a generational pattern $gen\_patt(i, j) = (a, b)$ an ordered pair (a, b) of two internal nodes from respectively first and second of compared trees that are (i, j)'s most recent common ancestors.

   We build a matrix m, where for a, b, $m(a, b)$ is the number of leaves' pairs for which (a, b) is a generational pattern. To speed up the algorithm, we add a row and a column with the sum in an appropriate line (see table 8.7).

3. Computing the distance between description elements.

   If:
   - a generational pattern (a, b) occurs $k_{1,2}$ times and moreover
   - a generational pattern (a, x), where x states for all the T_2's internal nodes occurs k_1 times
   - a generational pattern (y, b), where y states for all the T_1's internal nodes occurs k_2 times

   then the distance between description elements A, B formed on respectively internal nodes a from T_1, b from T_2 is

   $$h(A, B) = k_1 + k_2 - 2k_{1,2} \tag{8.12}$$

   In other words, the distance between two description elements build on two trees' internal nodes is the number of leaves' pairs that have the most recent common ancestor in just one of two internal nodes.

As the algorithm performs only $O(1)$ operations during iterations through arrays of at most n^2 size, the algorithm costs $O(n^2)$ time.

EXAMPLE

Having two binary rooted trees T_1, T_2 on figure 8.6, we:

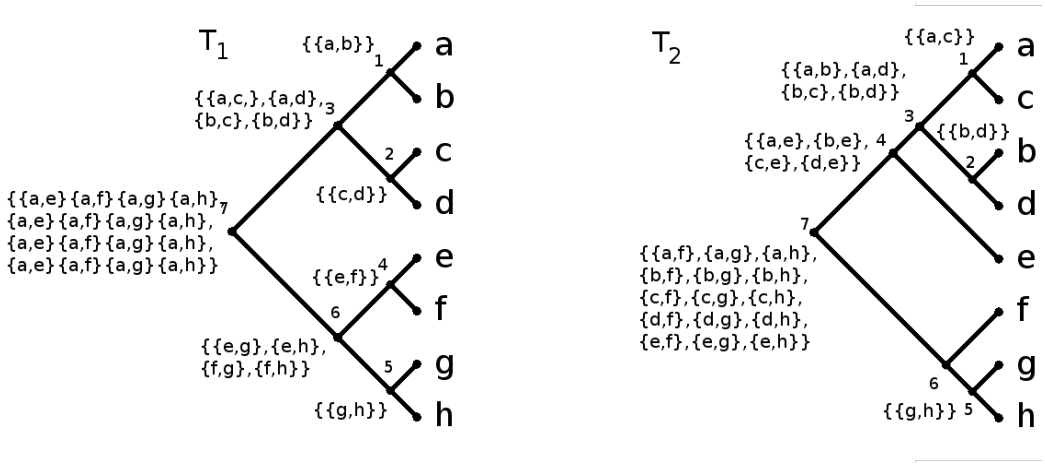


Figure 8.6: Two binary rooted trees. To each internal node is assigned a set of pairs of which the node is the most recent common ancestor.

1. For each tree - prepare a matrix with most recent common leaves pair' ancestor (tables 8.5 and 8.5).

2. Prepare a matrix with $T_1 - T_2$ generational patterns count (see table 8.7 and its description).

3. Calculate a cost matrix according to equation 8.12 - see table 8.8.

4. Compute the minimum weighted perfect matching on elements from table 8.8 and so acquire the pairs minimum weight perfect matching distance $mp(T_1, T_2) = 22$.

Leaf	a	b	c	d	e	f	g	h
a		1	3	3	7	7	7	7
b			3	3	7	7	7	7
c				2	7	7	7	7
d					7	7	7	7
e						4	6	6
f							6	6
g								5
h								

Table 8.5: T_1 matrix with pairs of leaves' most recent common ancestor.

Leaf	a	b	c	d	e	f	g	h
a		3	1	3	4	7	7	7
b			3	2	4	7	7	7
c				3	4	7	7	7
d					4	7	7	7
e						7	7	7
f							6	6
g								5
h								

Table 8.6: T_2 matrix with pairs of leaves' most recent common ancestor.

internal node $T_1 \setminus T_2$	*1*	*2*	*3*	*4*	*5*	*6*	*7*	count
1	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1
3	1	1	2	0	0	0	0	4
4	0	0	0	0	0	0	1	1
5	0	0	0	0	1	0	0	1
6	0	0	0	0	0	2	2	4
7	0	0	0	4	0	0	12	16
count	1	1	4	4	1	2	15	

Table 8.7: $T_1 - T_2$ internal nodes' generational pattern count; ei. T_1's internal node of id $= 7$ with T_2's internal node of id $= 4$ are the most recent common ancestors for 4 pairs of leaves. The last row and the last column count the total number of pairs that have an internal node as the most recent common ancestor.

Description element: internal node $T_1 \setminus T_2$	*1*	*2*	*3*	*4*	*5*	*6*	*7*
1	2	2	3	5	2	3	16
2	2	2	3	5	2	3	16
3	3	3	4	8	5	6	19
4	2	2	5	5	2	3	14
5	2	2	5	5	0	3	16
6	5	5	8	8	5	2	15
7	17	17	20	12	17	18	7

Table 8.8: The costs matrix; e.i. the cost between T_1's internal node of id $=$ 6 and T_2's internal node of id $= 7$ is $h(6, 7) = 4 + 15 - 2 \times 2 = 15$

8.5 The Hungarian algorithm – finding the minimum weight perfect matching

Finding a minimum weight perfect matching (the assignment problem) is an important aspect of the trees comparison algorithms from this chapter. Although it is an optimization problem, the Kuhn-Munkres theorem transforms it to combinatorial one and this combinatorial problem was a research object for a few surveys, which listing with results in polynomial algorithm solutions is presented in table 8.9.

	$O(nW \cdot \mathrm{VC}(n,m))$	Egerváry [1931] (implicitly)
	$O(2^n n^2)$	Easterfield [1946]
	$O(nW \cdot \mathrm{DC}(n,m,W))$	Robinson [1949]
	$O(n^4)$	Kuhn [1955b], Munkres [1957][28] Hungarian method
	$O(n^2 m)$	Iri [1960]
	$O(n^3)$	Dinits and Kronrod [1969]
*	$O(n \cdot \mathrm{SP}_+(n,m,W))$	Edmonds and Karp [1970], Tomizawa [1971]
	$O(n^{3/4} m \log W)$	Gabow [1983b,1985a,1985b]
*	$O(\sqrt{n}\, m \log(nW))$	Gabow and Tarjan [1988b,1989] (cf. Orlin and Ahuja [1992])
	$O(\sqrt{n}\, mW)$	Kao, Lam, Sung, and Ting [1999]
*	$O(\sqrt{n}\, mW \log_n(n^2/m))$	Kao, Lam, Sung, and Ting [2001]

Table 8.9: Complexity survey for the maximum(/minimum)-weight bipartite matching. * indicates an asymptotically best bound in the table. n - number of vertices, W - the maximum of the absolute of all the weights, m - number of edges, $VC(n,m)$ - the time required to find a minimum vertex cover in a bipartite graph, $DC(n,m,W)$ - the time to find a negative-lingth directed circuit in a directed graph with integer weight of arcs, each at most W in absolute value, SP_+ - the time needed to find a shortest path in a directed graph with non-negative length of edges each at most W. [31].

Among the other algorithms, there is the one developed in 1955 by the John von Neuman Theory Prize winner, professor of mathematics Harold Knuth. The mathematician named the algorithm after the nationality of the two mathematicians Dénes Kőnig and Jenő Egerváry, whose work was the base for The Hungarian Method. The original algorithm took $O(n^4)$ time, however

Edmonds and Karp, and independently Tomizawa noticed that it can be modified to achieve an $O(n^3)$ running time.

In this section we briefly introduce the $O(n^3)$ algorithm, basing mainly on the Hong Kong University of Science and Technology course notes [48].

TAGS
bipartite graph, complete graph, matching, perfect matching, minimum weight perfect matching, assignment problem

INPUT DATA
A complete bipartite graph with partitions of the same size.

THE $O(n^3)$ ALGORITHM
The algorithm discussed in this section does not find the requested for MWMP algorithms minimum weight perfect matching but a maximum weight perfect matching. Nevertheless, the transformation can be easily done by e.i. considering the additive inverse of an input edge weights.

The algorithm bases on the Kuhn-Munkres theorem that transforms an optimization problem to a combinatorial of finding a perfect matching. It combinatorializes the weights which is a classic technique in combinatorial optimization.

To present the theorem and the algorithm, we need to introduce a few terms (consult also figures 8.7 and 8.8).
Having a bipartite graph $G = (V, E)$ (not necessarily complete) with partitions X, Y: $|X| = |Y|$, weights $w(x, y)$ on edges $\{x, y\}$ $(x \in X, y \in Y)$ and M-any matching in G we introduce:

- an *alternating path* – a path in the graph G of which edges alternate between M and $E \setminus M$.
- an *augumenting path* – an alternating path of which both endpoint edges are not in M.
- an *alternating tree* – a tree built on some edges from G, rooted at some G's vertex v such that $\neg\exists_{y \in V}\{v, y\} \in M$. Every path from v in an alternating tree is an alternating path.
- a *feasible labelling* – a function $l : V \to R$ such that $\forall_{x \in X, y \in Y} l(x) + l(y) \geq w(x, y)$, where $w(i, j)$ is a weight of the edge between i and j.
- an *equality graph* – a subgraph of G with respect to l so that $G_l = (V, E_l)$ where $E_l = \{\{x, y\} : l(x) + l(y) = w(x, y)\}$

The algorithm described in this section aims at finding a feasible labelling such that the equality graph built basing on the labelling contains a perfect

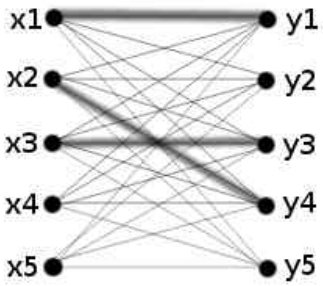


Figure 8.7: A complete bipartite graph. The shadowed edges belong to matching M. We can notice:
- alternating paths, e.i. $x_3 - y_3 - x_4, y_2 - x_3, y_2 - x_2 - y_4$. The longest consist of 7 edges, e.g. $x_5 - y_4 - x_2 - y_2 - x_1 - y_1 - x_3 - y_3$.
- augmenting paths, e.i. $x_4 - y_3 - x_3 - y_2, x_5 - y_4 - x_2 - y_2 - x_1 - y_1 - x_3 - y_3$.
- alternating trees, e.i. root at $x4$ and paths $x_4 - y_3 - x_3 - y_2 - x_2 - y_4, x_4 - y_1 - x_1 - y_5$.

Figure 8.8: On the left: A graph with feasible labelling l. On the right: An equality graph G_l.

matching. This matching is the maximum weight for a graph G according to the Kuhn-Munkres theorem.

Kuhn-Munkres theorem. *If l is feasible and M is a perfect matching in E_l then M is a maximum weight matching.*

Proof. Denote edge $e \in E$ by $e = \{x, y\}$. Let M' be any perfect matching in G (not necessarily in in E_l) and $w(M')$ - the weight of M'. Since every $v \in V$ is covered exactly once by M' we have $w(M') = \sum_{e \in M'} w(e) \leq \sum_{\{x,y\} \in M'} (l(x) + l(y)) = \sum_{v \in V} l(v)$ so $\sum_{v \in V} l(v)$ is an upper-bound on the cost of any perfect matching.

Now let M be a perfect matching in E_l. Then $w(M) = \sum_{e \in M} w(e) = \sum_{v \in V} l(v)$. So $w(M') \leq w(M)$ and M is optimal.

□

The idea of the algorithm is to:

- start at some feasible labelling l and some matching M in E_l

- while M is not perfect repeat the following:
  1. Find an augmenting path for M in E_l; this increases the size of M.
  2. If no augmenting path exists, improve l to l' such that $E_l \subset E_{l'}$. Go to 1.

Corollary 1. *The algorithm above stops and this happens when a maximum weight perfect matching is found.*

Proof. In each step of the loop we either increase the size of M or E_l so this process must terminate.

Furthermore, when the process terminates, M is a perfect matching in E_l for some feasible labeling l, so by the Kuhn-Munkres theorem, M is a maximum weight matching.

□

The improving l function and its properties are presented in the lemma below:

Having a bipartite graph $G(V, E)$ with two equal size partitions X, Y ($X \cup Y = V, X \cap Y = 0$), we

- denote l and l' as feasible labellings in G,
- define neighbourhood of $u \in V$ to be $N_l(u) = v : \{u, v\} \in E_l$, and neighbourhood of $S \subseteq V$ to be $N_l(S) = \bigcup_{u \in S} N_l(u)$

Let $S \subseteq X$ and $T \subset Y : T = N_l(S) \neq Y$.

Lemma. *If we set:*
$\alpha_l = min_{x \in S, y \notin T}\{l(x) + l(y) - w(x, y)\}$

$$l'(v) = \begin{cases} l(v) - \alpha_l & : v \in S \\ l(v) + \alpha_l & : v \in T \\ l(v) & : otherwise \end{cases}$$

Then l' is a feasible labelling and for $x \in X, y \in Y$.

- *If $\{x, y\} \in E_l$ for $x \in S, y \in T$ then $\{x, y\} \in E_{l'}$.*
- *If $\{x, y\} \in E_l$ for $x \notin S, y \notin T$ then $\{x, y\} \in E_{l'}$.*
- *There is some edge $\{x, y\}$ that $\{x, y\} \in E_{l'}$ and $x \in S, y \notin T$.*

Proof. The operation that improves the labelling l to l' basically decrements all the labels of $v \in S \subseteq X$ by the same value that it increments all the labels of $u \in T \subset Y$.

This does not change the feasible labelling idea that $\forall_{x \in X, y \in Y} l(x) + l(y) \geq w(x, y)$ and allows $\forall_{\{x,y\} \in E_l, x \in S, y \in T} \{x, y\} \in E_{l'}$ and $\forall_{\{x,y\} \in E_l, x \in X \setminus S, y \in Y \setminus T} \{x, y\} \in E_{l'}$ because:

- $\forall_{x \in S, y \in T} l'(x) + l'(y) = l(x) + l(y)$,
- $\forall_{x \in X \setminus S, y \in Y \setminus T} l'(x) + l'(y) = l(x) + l(y)$
- $\forall_{x \in X \setminus S, y \in T} l'(x) + l'(y) = l(x) + l(y) + \alpha > l(x) + l(y)$
- $\forall_{x \in S, y \in Y \setminus T} l'(x) + l'(y) = l(x) + l(y) - \alpha$. As $\alpha \leq l(x) + l(y) - w(x, y)$, therefore $l(x) + l(y) - \alpha \geq l(x) + l(y) - (l(x) + l(y) - w(x, y)) = w(x, y)$

Moreover, there is (at least one) $y \in Y \setminus T$ which together with $x \in S$ generated the value of $\alpha = l(x) + l(y) - w(x, y)$, therefore $l'(x) + l'(y) = l(x) + l(y) - (l(x) + l(y) - w(x, y)) = w(x, y)$ and so $\{x, y\} \in E_{l'}$.

□

Now we present more detailed proceedings for the $O(n^3)$ Hungarian Algorithm. Having a graph $G = (V, E)$

1. Generate initial labelling l and matching M in E_l.

2. If $|M| = |V|/2$ (so if M is perfect) then:
   return M.
   Otherwise:
   pick a vertex $u \in X : \neg\exists_{y \in Y} \{u, y\} \in M$,
   set $S = \{u\}, T = \emptyset$.

3. If $N_l(S) = T$ then:
   improve labelling (forcing $N_l(S) \neq T$)

4. Pick $y \in N_l(S) \setminus T$.
   If $\neg\exists_{x \in X} \{x, y\} \in M$ (the path from u to y is an augmenting path) then:
   augment M (for details, see description of this step below),
   go to step 2.
   Otherwise:
   pick $x \in X$ such that $\{x, y\} \in M$
   extend alternating tree: $S = S \cup \{x\}, T = T \cup \{y\}$.
   go to step 3.

We give some more detailed description of the algorithm steps:

Step 1:
Generating an initial labelling and matching may be proceeded as follows:
$M = \emptyset$
$\forall_{y \in Y} l(y) = 0$
$\forall_{x \in X}$:
$l(x) = max_{y \in Y} w(x, y)$
$z = argmax_{y \in Y} w(x, y)$
`if (`z `is not matched by` M`) then`
$M = M \cup \{\{x, z\}\}$.

Step 4:
Consider a part of graph G pictured in the figure 8.9. Edges $\{x_k, y_i\}$ and $\{x_j, y_j\}$ belong to a matching M, other vertices in the picture - x_i and y_k do not form edges that belong to M. Assume that we are in step 2 and $S = \{x_i\}, T = \emptyset$.
As $l(x_i) + l(y_j) = w(x_i, y_j), l(x_i) + l(y_k) = w(x_i, y_k)$, but $l(x_i) + l(y_i) \neq w(x_i, y_i)$, therefore $y_j, y_k \in N_l(S) \setminus T$, but $y_i \notin N_l(S) \setminus T$.

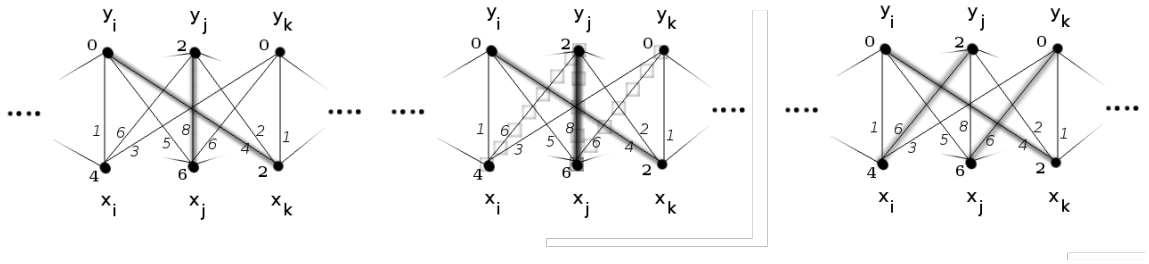


Figure 8.9: A part of graph $G(V, E)$.
On the left: higlighted $\{x_k, y_i\}$, $\{x_j, y_j\} \in M$.
In the middle: alternating tree rooted at x_i that consists only one path $x_i - y_j - x_j - y_k$ is marked with a squared line.
On the right: new matching is higlighted.

Now we enter step 4, picking $y = y_j$. As $\{x_j, y_j\} \in M$, we extend alternating tree e.g. $S = S \cup \{x_j\}, T = T \cup \{y_j\}$. The perfect matching is not found (step 2) and $N_l(S) \neq T$ (step 3); out of y_i, y_j, y_k, only $y_k \in N_l(S) \setminus T$. Lets pick the $y = y_k$ (step 4). As there is no $x \in X$ such that $\{x, y\} \in M$, there is an augmenting path $x_i - y_k - x_k - y_j$ found. Now we augment M in a following way:

Edges from the augmenting path that were in the M are excluded form M and edges from the augmenting path that were not in the M are added to M. As in the augmenting path the number edges $e \notin M$ is one greater that the number of $e \in M$, the size of M is increased by 1.
Now (step 2) if all the other G vertices are part of edges $e \in M$, the perfect matching is found. Otherwise, there is chosen a vertex $u \in X, \neg\exists_{y \in Y}\{u, y\} \in M$ and of course $u \notin \{x_i, x_j, x_k\}$. u is a root of a new alternating tree that will help in finding next augmenting path.

CORRECTNESS AND COMPLEXITY

We denote a *phase* of an algorithm to be the set of operations that starts with setting $S = \{u\}, T = \emptyset$ and ends with incrementing M by 1. Each phase consists of iterations where by *iteration* we mean the optional step 3 (labels improving) and the obligatory step 4 which is either augmenting M and ending the phase or extending an alternating tree.

In any iteration, if $N_l(S) = T$, we improve labels. As the lemma guarantees that : $\forall_{\{x,y\} \in E_l, x \in X \setminus S, y \in Y \setminus T}\{x, y\} \in E_{l'}$ the current M remains in equality graph. Similarly due to $\forall_{\{x,y\} \in E_l, x \in S, y \in T}\{x, y\} \in E_{l'}$ the edges constituting the alternating tree built so far remain in equality graph.

Picking $y \in N_l(S) \setminus T$, if we can create an augmenting path, then the size of M is increased. Otherwise there is $x : \{x, y\} \in M$ and we augment an alternating tree. The tree can have at most $|M|$ number of $y \in Y$ vertices and as is this occurres, the next iteration must augment M.

This shows that the algorithm always terminates and when it does, according to the Kuhn-Munkres theorem, M is a perfect matching in E_l so, a maximum weight perfect matching in G. □

The number of phases can be at most $|V|/2 - 1$, depending on the initial matching size, which can minimally be equal to 1. As for an iteration, in each one there is either M augmented which ends a phase or alternating tree is extended. This way the maximum number of iterations is the maximal possible size of an alternating tree before there is no $y \in Y : \exists_{x \in X}\{x, y\} \in M$ which is $O(|V|)$.

The steps 3 and 4 are therefore performed $O(|V|^2)$ times. We can achieve the $O(|V|)$ time for internal procedures of each of those steps by keeping for each node $y \in Y$ a track $slack_y = argmin_{x \in S}(l(x) + l(y) - w(x, y))$. Initialization of the slack array is performed at the beginning of each phase and takes $O(|V|)$ time as there is only one node in the S set. In step 4 when a vertex

$x \in X$ is added to S, all the slacks must be scanned and updated if the value for new S element is less than current; this takes $O(|V|)$ time. The goal of the slack array is that it allows to improve labelling (choose the α value) in $O(|V|)$ time, as $\alpha = min_{y \in T} slack_y$. After calculating α, as the labels of all the $v \in S$ changed, all the slacks must be updated, which can be done in $O(|V|)$ by setting $\forall_{y \in T} slack_y = slack_y - \alpha$.

The step 1 is performed once and requires $O(|V|^2)$ time. The step 2 is only performed at the beginning of each of $O(|V|)$ phases. Keeping a flag for each $x \in X$ that informs whether a vertex is a part of edge that belongs to M allows to perform operations of a single step 2 in $O(|V|)$ time.

Thus the total time complexity of Hungarian Algorithm is $O(|V|^3)$ and as all the auxiliary structures require $O(|V|)$ space, the space complexity is $O(|V|)$.

Part III

The PhylotreeDist library

Beksiński, 1979, [102]

The best time to plant a tree is twenty years ago. The second best time is now.

African proverb

Introduction

In this part we focus on implementation of our C++ trees distances library. We discuss its functionalities, construction, implementation and usage, also giving a library's manual.

In order to have a better point of view on the library, we also present a comparison existing phylogenetic libraries in two contexts: libraries that contains phylogenetic trees distances and popular C++ phylogenetic libraries.

Chapter 9

Existing Solutions

In order to get better understanding of the field of phylogenetics in computer science, we survey the existing phylogenetic libraries. We consider the libraries in two aspects that concern our library:

- libraries that provide trees distance methods
- popular C++ bioinformatic libraries (not necessarily concerning the trees distance methods)

For each library we provide information on a number and date of their last release in order to give an outline whether the software is currently developed. The research is done on February 2011.

9.1 The libraries that provide trees distance methods

The Felsenstein/Kuhner lab in the Department of Genome Sciences at University of Washington, Seattle claims to present the largest phylogeny programs list anywhere in the web [86]. The listing's content is regularly updated which is noticeable by a long list of recent changes. The listing is cited by multiple other sources, also articles and books e.i. [26], [35] or [24]. For all that purposes the site is the basic also for our researches. Moreover, the site contains a special section where there is a list of programs that have functionalities to obtain consensus trees, subtrees, supertrees, and distances between trees - which is our main interest area.

As we are focused on programs that computes trees distances, we examine the documentations of the programs from the list to see what metrics they provide. The review of Phylogenetic libraries that offer trees distance given in table 9.1 aims at presenting those libraries in context of the language they are written in, whether they are available free of charge, whether their source code is open and whether they are actively mantained.
The listing is sorted alphabetically by language of source code.

library name	free of charge	open source	last release	lang.	provided trees distances	remarks
BIRCH [49]	Y	Y	V2.7 2010-05	C	RF	
PAUP [65]	N	N	V4.0 2002	C	RF(1), two agreement subtree metrics(2)	(1) is called „Symmetric difference". For (2) description, see [65]
PHYLIP – TREEDIST program [68]	Y	Y	V3.69 2009-09	C	RF, Branch Score distance (weighted RF)	It has been distributed since October, 1980 and has celebrated its 30th anniversary as the oldest distributed phylogeny package
RAxML [73]	Y	Y	V7.2.6 2010-03	C	RF, weighted RF	
Robinson and Foulds topological distance [74]	Y	Y	2000-07-17	C	RF	

EMBOSS [55]	Y	Y	V6.3.1 2010-07	C, also C++, Java, Perl	RF, Branch Score Distance(2)	(2) is weighted RF. For details, see [56]
Bio++ [40]	Y	Y	V2.0 2011-02-08	C++	RF	
Murka [62]	Y	Y	V1.3.1 2011-01-29	C++	RF, Pairwise weighted tree distance (1)	For (2) description see Murka documentation
QuartetSuite [72]	Y	Y	V1.0 2003-06	C++	Quartet	
PhyloNet [70]	Y	Y	V2.3 2010-11	Java	RF, distance between two networks	
Mesquite [61]	Y	Y	V2.74 2010-10-03	Java	RF, visualizations of tree space	
COMPONENT [53]	Y	Y	V2.0 2001-08	Pascal	RF for rooted and unorooted trees(1), NNI metric, triplets, quartets, agreement subtrees	Description of the methods is available at [54]. (1) is also called a 'partition metric'
Supertree package – patririon metric [77]	Y	Y	v.1.2.1 2007-09-14	Perl	RF	RF is called „partition metric"
TopD/fMts [78]	Y	Y	V3.3 24-08-2007	Perl, TclTk	RF, Nodal, Quartet, Triplets, TOPD, FTMS	For details about TOPD, FTMS see [78]

Phangorn [66]	Y	Y	V1.3-1 2011-02-14	R	RF(1), Branch Score Distance(2), Nodal(3), weighted Nodal.	(1) is called „Symmetric difference". (2) is weighted RF (3) is called a „path difference".
Phybase [67]	Y	Y	V1.1 2010-08-05	R	RF	
Clann [52]	Y	N	V3.0.2 2005-11-27	(No info published)	Most Similar Supertee Method, Quartet distance (2), RF (3)	Algorithms described in the manual. They aim at comparing trees with supertree. (2) is called Maximum Quartet Fit, (3) is called Maximum Split Fit
PhyNav [71]	Y	N	V 1.0 2004-11	(No info published)	The shortest triplet clustering (STC)	The program finds subsets of species in a dataset that are "minimal k-distance subsets" and analyses these each by maximum likelihood. Then it stitches these groups together using likelihood. The algorith is described in details in [71]

Table 9.1: The phylogenetic libraries in context of trees distance they provide. The table contains also information about whether the library is available free of charge, whether the source code is open, the date and number of the last release, the language of code and. Sorted alphabetically by language of source code. (RF stands for Robinson-Foulds).

9.2 Popular C++ bioinformatic libraries review

Fourment and M.R. Gillings in the article [21] have presented benchmarks for common programming languages used in bioinformatics. They compared the memory usage and speed of execution for three standard bioinformatics methods: Sellers algorithm, the Neighbor-Joining tree construction algorithm and an algorithm for parsing BLAST file outputs; implemented in programs using one of six different programming languages: C, C++, C#, Java, Perl

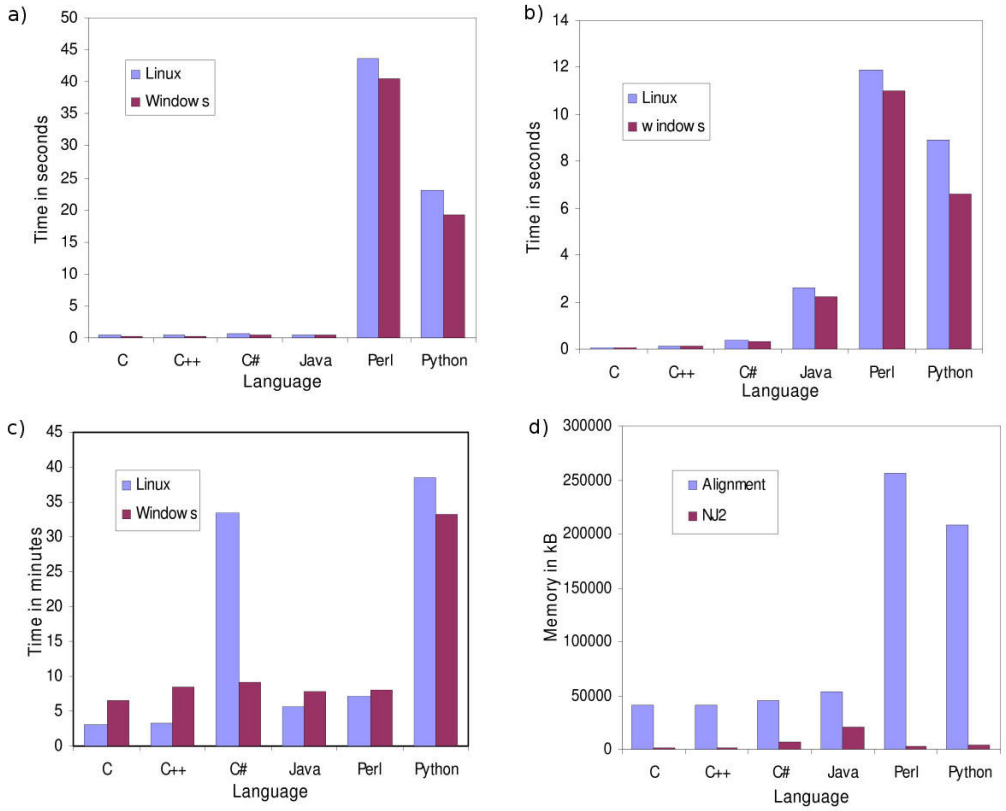


Figure 9.1: Speed and memory comparison for C, C++, C#, Java, Perl and Python languages. [21]
a) Speed comparison of the global alignment program.
b) Speed comparison of the Neighbor-Joining program.
c) Speed comparison of the BLAST parsing program.
d) Memory usage comparison of the Neighbor-Joining and global alignment programs.

and Python.

The results were compatible with popular attitudes and showed that implementations in C and C++ were fastest and used the least memory (see fig. 9.1) though generally contained more lines of code than others. Java and C# were said to be a compromise between the flexibility of Perl and Python and the fast performance of C and C++, however their results did not drastically differ from those of C and C++. The relative performance of the tested languages did not change from Windows to Linux and no clear evidence of a faster operating system was found. As a result, just as it is generally considered, the best choice of language for a task would be according to needs, keeping in mind that Java is a portable web oriented language, Perl is a powerful script language, Python is an easily coded language and C and C++ are efficient languages used in operating systems and drivers.

As our PhylotreeDist library, which aims at having good memory and time performance, is written in C++, below we present the most popular C++ libraries that concern phylogeny issue: the EMBOSS, NCBI C++ Toolkit, Bio++ and PHYLIP libraries. For each of them, we outline its general specification (tables 9.2,9.3,9.4,9.5), quote abstract from its homepage, describe the library in the phylogeny context and in the trees distances context and give a comment that collects and summarizes different information about a library.

EMBOSS

Organisation	Licence	Supported operating systems	Last release	References	Provided trees metrics
OBF - Open Bioinformatics Foundation	GNU	Linux, SUN Solaris, Windows etc	V6.3.1 2010-07	[55] [56] [57] [58]	RF, Branch Score Distance

Table 9.2: EMBOSS specification.

ABSTRACT FROM THE LIBRARY'S HOMEPAGE [55]
The acronym EMBOSS stands for "The European Molecular Biology Open Software Suite". EMBOSS is a free Open Source software analysis package specially developed for the needs of the molecular biology (e.g. EMBnet) user

community. The software automatically copes with data in a variety of formats and even allows transparent retrieval of sequence data from the web. Also, as extensive libraries are provided with the package, it is a platform to allow other scientists to develop and release software in true open source spirit. EMBOSS also integrates a range of currently available packages and tools for sequence analysis into a seamless whole. EMBOSS breaks the historical trend towards commercial software packages.

PHYLOGENY CONTENT

EMBOSS contains a large applications groups listing where 7 out of 45 (as for the year 2011) are phylogenetics oriented and are as follows:

- Phylogeny consensus
- Phylogeny continuous characters
- Phylogeny discrete characters
- Phylogeny distance matrix
- Phylogeny gene frequencies
- Phylogeny molecular sequence
- Phylogeny tree drawing

Some phylogeny programs do not strictly belong to the EMBOSS suite, but are a part of the PHYLIP from the EMBASSY package. EMBASSY is the associated software that includes applications with the same look and feel as EMBOSS applications, but which authors wish to be kept separate from EMBOSS. In the case of the PHYLIP package, the programs were ported from Joe Felsenstein's PHYLIP package, version 3.61 (August 2004). Next PHYLIP was converted to PHYLIPNEW which since the EMBOSS 3.0.0 is released with the EMBOSS suite. The PHYLIPNEW versions of PHYLIP programs all have the prefix "f" to distinguish them from the original programs.

TREES DISTANCES CONTENT

The Phylogeny consensus group contains a ftreedist program that computes the Branch Score distance between trees and the Robinson-Foulds symmetric difference distance between trees. The comprehensive description of the program including the Branch Score distance algorithm and sample usage, sample output is available at [57].

COMMENT
The EMBOSS library is one of the main projects of Open Bioinformatics Foundation – a major organisation that focuses on supporting open source programming in bioinformatics. EMBOSS is a suite containing wide set of programs. It is written mostly in C, but contains also C++, Java or Perl code. The Phylogenetics part of the library is generally ported from the PHYLIP package – one of the most popular and probably the oldest distributed phylogeny package (the more detailed PHYLIP description later in this chapter). The two trees distances that are provided by the library as a part of PHYLIPNEW library are the Robinson-Foulds and the Branch Score Distance. They are both written in C.

NCBI C++ Toolkit

Organisation	Licence	Supported operating systems	Last release	References	Provided trees metrics
NCBI - National Center for Biotechnology Information (USA)	Free, portable, public domain libraries with no restrictions use	Unix, MacOS, Windows	2010-06	[63]	—

Table 9.3: NCBI specification.

ABSTRACT FROM THE LIBRARY'S HOMEPAGE [63]
The NCBI C++ Toolkit provides a lot of useful libraries, both general purpose and biotech-related that are constantly developed, maintained and used in real-life production by hundreds of Web and standalone applications and their programmers (also counted in hundreds).

If you are a C++ developer you will find the portable nature of the libraries very useful in building cross-platform applications even if you do not have much interest in Bioinformatics. Libraries such as the ones for the CGI/Fast-CGi, HTML, Networking, SQL Database Access, ASN.1 and XML Serialization are quite general purpose and can be used in a variety of applications outside the Bioinformatics problem domain. The C++ Toolkit undergoes active development with the libraries being built every night. The

documentation for the C++ Toolkit is available online in theNCBI Bookshelf format and also as downloadable book in Acrobat's PDF format.

PHYLOGENY CONTENT
The phylogenetics algorithms are included in the algo/phy_tree section of the ALGORITHM module(algo/phy_tree directory).

TREES DISTANCES CONTENT
The NCBI library do not provide any methods for computing trees distances.

COMMENT
The NCBI library is a "United States Government Work", at it states in the notice of the software. It was written as part of authors offical duties as a United States Government employees. The phylogenetic part of NCBI library is not much compex, in particular there are no algirithms that operate on trees topology.

Bio++

Organisation	Licence	Supported operating systems	Last release	References	Provided trees metrics
Personal work of Julien Dutheil, Sylvain Gaillard et.al from Université Montpellier, France	CeCILL (code free of charge)	Linux, MacOS, Unix, Windows (using the Cygwin port and MinGW)	V2.0 2011-02-08	[40] [20]	RF

Table 9.4: Bio++ specification.

ABSTRACT FROM THE LIBRARY'S HOMEPAGE [40]
Bio++ is a set of C++ libraries for Bioinformatics, including sequence analysis, phylogenetics, molecular evolution and population genetics. Bio++ is fully Object Oriented and is designed to be both easy to use and computer efficient.

Available components include classes for data storage and handling (nucleotide/amino-acid/codon sequences, trees, distance matrices, population genetics datasets), various input/output formats, basic sequence manipulation (concatenation,

transcription, translation, etc.), phylogenetic analysis (maximum parsimony, markov models, distance methods, likelihood computation and maximization), population genetics/genomics (diversity statistics, neutrality tests, various multi-locus analyses) and various algorithms for numerical calculus.

We did a collaborative effort to design a hierarchy of useful classes for the rapid development of efficient applications in the fields of sequence analysis, phylogenetics, molecular evolution and population genetics. These libraries combine tools for data handling and numerical calculations, providing an easy-to-use, powerful and general development environment. The three main objectives of Bio++ are (i) to speed up the implementation of a new method in order to test it, (ii) to make easier the development of robust applications for distribution and (iii) to provide a high level programming language for biologists.

PHYLOGENY CONTENT
Bio++ classes are split into five libraries: three main biological libraries (SeqLib, PhylLib and PopGen-Lib) and two utility libraries (Utils and Num-Calc). The library is more focused on molecular evolution (model fitting, ancestral state reconstruction, across-site rate variation) than pure phylogenetic reconstruction, although several topology reconstruction methods are implemented (distance methods, nearest-neighbor interchanges (NNI) search for maximum likelihood (ML) and parsimony scores).

TREES DISTANCES CONTENT
The PhyLib library provides a *bpp::TreeTools* class that contains static utilitary methods which deal with trees. This includes the *bpp::TreeTools::robinsonFouldsDistance* method, that works in $O(n^3)$ time.

COMMENT
The Bio++ package is well documnted (in doxygen). Dutheil et al. in [20] describe it as comprehensive library providing re-usable components for phylogenetics, molecular evolution, and population genetics; a purely object-oriented library, favoring ease of development over performance and scalability. The phylogeny package has an intuitive, usable interface. The library is actively developed, can be noticed as new major version 2.0 was released in February 2011.

Organisation	Licence	Supported operating systems	Last release	References	Provided trees metrics
Personal work of Joseph Felsenstein , proffesor at Department of Genome Sciences and the Department of Biology at the University of Washington, Seattle.	Permission is granted to copy and use this program provided no fee is charged for it and provided that this copyright notice is not removed.	Linux, MacOS, Windows	V3.69 2009-09	[68]	Robinson-Foulds, Branch Score distance

Table 9.5: PHYLIP specification.

PHYLIP

ABSTRACT FROM THE LIBRARY'S HOMEPAGE [68]
PHYLIP (the PHYLogeny Inference Package) is a package of programs for inferring phylogenies (evolutionary trees). Methods that are available in the package include parsimony, distance matrix, and likelihood methods, including bootstrapping and consensus trees. Data types that can be handled include molecular sequences, gene frequencies, restriction sites and fragments, distance matrices, and discrete characters.

The programs are controlled through a menu, which asks the users which options they want to set, and allows them to start the computation. The data are read into the program from a text file, which the user can prepare using any word processor or text editor. At this stage we do not have a mouse-windows interface for PHYLIP.

PHYLIP is probably the most widely-distributed phylogeny package. It is the third most frequently cited phylogeny package, after PAUP and MrBayes, and ahead of MEGA. PHYLIP is also the oldest widely-distributed package. It has been in distribution since October, 1980, and has over 28,000 registered users. It is regularly updated.*

PHYLOGENY CONTENT
The whole library is a phylogenetics package. In particular, the library deals with parsimony, distance matrix, and likelihood methods, including bootstrapping and consensus trees. As for the tree distances, it includes Robinson-Foulds and Branch Score distance algorithms.

TREES DISTANCES CONTENT
The distance methods are available through *treedist* program that computes the Branch Score distance between trees and the Robinson-Foulds symmetric difference distance between trees.

COMMENT
The PHYLIP package is the oldest and the most and widely used set of phylogenetic programs: [69] is claims PHYLIP to have about 29,000 registered users. It has a big historical influence on phylogenetics in computer science: PHYLIP has been distributed since October, 1980 and in 2010 celebrated its 30th anniversary, as the oldest distributed phylogeny package. The library is currently developed as an individual package and also converted and kept as PHYLIPNEW, one of the EMBOSS library package.

Other C++ bioinformatics libraries

We have also considered a few other C++ bioinformatics libraries that we found during researches. However, we did not present their detailed description as they do not include phylogenetics or were not available. Those are:

- BTL - The Bioinformatics Template Library [50], [51].
- libsequence - C++ class library for population genetics [60].
- SeqAn - C++ library of efficient algorithms and data structures for the analysis of sequences with the focus on biological data [75], [76].
- libcov – the paper describing the library [59] looks promising for phylogenetics purposes, but unfortunately neither the link to library page provided in the paper works nor Internet researches shows resources where the library (which in on the GPL license) can be downloaded from.

Conclusions

Searching the Internet, we came across a great variety of bioinformatic and in particular – phylogenetic programs. Having reviewed the issue of which distances between trees are implemented, we find noticeable that the most popular and often the only implemented metric is the Roubinson-Fould distance. Some libraries (Phangorn (in R), TOPD/FMTS (in Perl, TclTk), Clann (the language is not published) or Component (in Pascal)) provide more complex set of trees distance methods. However none of those is written in C++ which is the object of our interest.

Considering popular C++/C Bioinformatic libraries, four out of eight has phylogenetic package Out of those, just Bio++, EMBOSS and PHYLIP offer trees distances: all offer the Robinson-Foulds and EMBOSS and PHYLIP implement the Branch Score Distance.

Chapter 10

PhylotreeDist as a Plugin to Bio++ Phylogenetic Library

To implement tree distance metrics library we need

- data structures to store trees
- procedures to read the tree topology data and create a tree
- procedures to manipulate data, which perform actions in time enough low to obtain time complexity assumed by particular algorithm

These can be provided by external library or implemented. As there already exist several C++ phylogenetic libraries, in order to not reinvent the wheel and to popularize PhylotreeDist library in the same time, we decided to create the distance library that would be compatible with existing solution and would take advantage of the solution's achievements.

10.1 Reasons for the interest in Bio++

In the chapter 9 we conduct a research on existing bioinformatic libraries solutions. In this chapter we give a special attention to the Bio++ library, as the one that, after the research, we found the most appropriate for the purpose of our PhylotreeDist library: the written in C++, Bio++ library claims to focus on being efficient and written in an Object Oriented manner, released as Open-Source and actively developed (the recent as for the year 2011 and also the major 2.0 release was in 08 February 2011). Those attributes were mostly respected during the C++ libraries analyses.

Moreover, the library's authors claim that "Any contribution will be welcome, as specific functions or as additional libraries compatible with the present ones" [20], which also is an important matter as we aim at popularising the work, especially the Minimum Weight Perfect Matching distance approach.

Furthermore, the phylogenetics is one of four major Bio++ issues among sequence analysis, molecular evolution and population genetics and Bio++ provides a good documentation [37] and intuitive interface.

All the issues above together with the personal feeling of comfortableness in using Bio++ library resulted in the decision of adapting Bio++ as a phylogenetic base for the PhylotreeDist library.

10.2 General information

According to the Bio++ site [40], *"Bio++ is a set of C++ libraries for Bioinformatics, including sequence analysis, phylogenetics, molecular evolution and population genetics. Bio++ is fully Object Oriented and is designed to be both easy to use and computer efficient."*

Bio++ library is available by compilation from the sources and also: for Linux - by installation with RPM or Debian package managers; for MacOS - using pre-compiled set of libraries. The library is released under the CeCILL[1] licence. The code is well documented which gives a source for the generated with Doxygen[2] documentation that is also available online on [37]. For further information and consultation, the user can visit the Wiki at [41], the Bio++ Usage Help Forum at [39] or the Bio++ Development Forum at [36].

Bio++ classes are split into five sub-libraries: three main biological libraries (SeqLib, PhylLib and PopGen-Lib) and two utility libraries (Utils and NumCalc). The table 10.1 shows brief description and dependencies between the sub-libraries [20].

[1]CeCILL - a french acronym for Ce(A)C(nrs)I(NRIA)L(ogiciel)L(ibre), the French Free Software license, compatible with the GNU GPL1. [43]

[2]Doxygen = a documentation system for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors), Fortran, VHDL, PHP, C#, and to some extent D. [45]

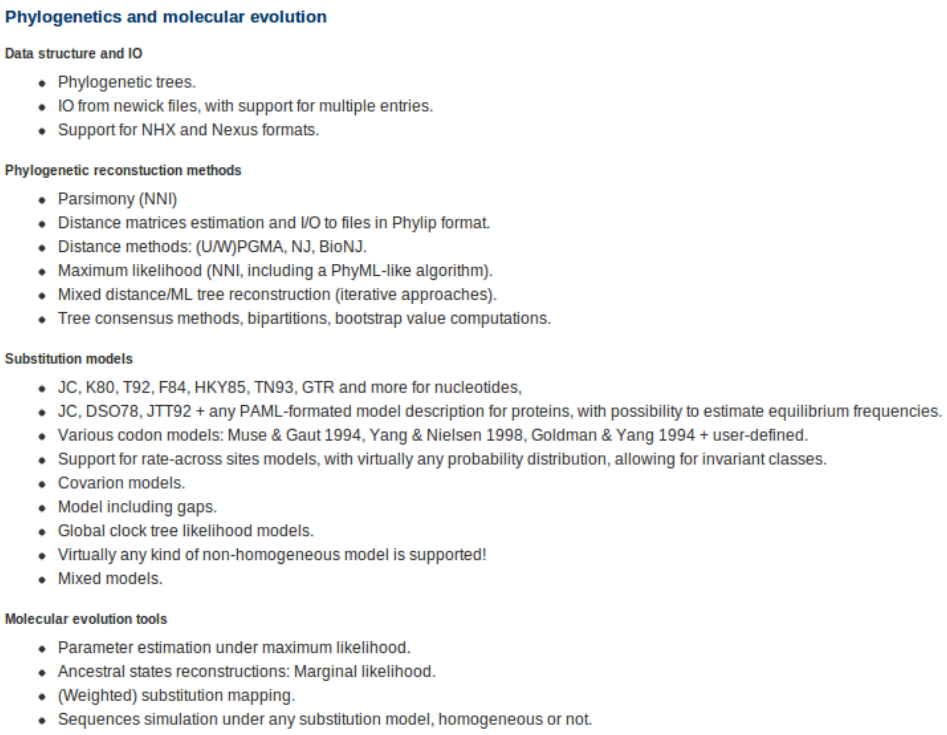
Sub-library	Description	Dependencies		
		Utils	NumCalc	SeqLib
Utils	Basal classes and utilities. Exceptions, text and file manipulation.		-	-
NumCalc	Numerical calculus. Vector and matrix manipulation, optimization, algebra, probability distribution and statistics, random number generation.	Y		-
SeqLib	Sequence manipulation, alphabets, chemical properties and distances, input/output in various formats (Fasta, Phylip, Clustal, etc.)	Y	Y	
PhylLib	Phylogenetics. Tree reconstruction by maximum parsimony, distance methods or maximum likelihood, parameter estimations, ancestral states reconstruction, etc.	Y	Y	Y
PopGenLib	Population genetics. Polymorphism statistics, linkage desequilibrium, neutrality tests and recombination.	Y	Y	Y

Table 10.1: The Bio++ library is a package of five sub-libraries. The table shows their description and dependencies. The "-" and "Y" symbols inform whether the sublibrary in a row is respectively independent and dependant on the sub-library in the column.

10.3 PhylLib – a description of the Bio++ phylogenetic package

The *PhylLib* is the name of the Bio++ phylogenetic package. In the figure 10.1 we present a listing from [40] with PhylLib functionalities available in Bio++ version 2.0.

The PhylotreeDist trees distances library takes advantage of the following PhylLib functionalities: input and output operations on Newick trees format files, the tree container and trees basic operations, the bipartition methods. For the tests we also compared output data of our Robinson-Foulds $O(n)$ implementation in PhylotreeDist to the $O(n^3)$ Bio++ algorithm's .



Phylogenetics and molecular evolution

Data structure and IO

- Phylogenetic trees.
- IO from newick files, with support for multiple entries.
- Support for NHX and Nexus formats.

Phylogenetic reconstuction methods

- Parsimony (NNI)
- Distance matrices estimation and I/O to files in Phylip format.
- Distance methods: (U/W)PGMA, NJ, BioNJ.
- Maximum likelihood (NNI, including a PhyML-like algorithm).
- Mixed distance/ML tree reconstruction (iterative approaches).
- Tree consensus methods, bipartitions, bootstrap value computations.

Substitution models

- JC, K80, T92, F84, HKY85, TN93, GTR and more for nucleotides,
- JC, DSO78, JTT92 + any PAML-formated model description for proteins, with possibility to estimate equilibrium frequencies.
- Various codon models: Muse & Gaut 1994, Yang & Nielsen 1998, Goldman & Yang 1994 + user-defined.
- Support for rate-across sites models, with virtually any probability distribution, allowing for invariant classes.
- Covarion models.
- Model including gaps.
- Global clock tree likelihood models.
- Virtually any kind of non-homogeneous model is supported!
- Mixed models.

Molecular evolution tools

- Parameter estimation under maximum likelihood.
- Ancestral states reconstructions: Marginal likelihood.
- (Weighted) substitution mapping.
- Sequences simulation under any substitution model, homogeneous or not.

Figure 10.1: An outline of PhylLib functionalities available in Bio++ version 2.0 from Bio++ site [40].

The Newick trees format is a popular in phylogenetic libraries format to store phylogenetic trees. Its name comes from the Newick's restaurant in Dover, New Hampshire, US, where in 1986 the notation's developers were discussing its format. The format allows to represent both rooted and unrooted trees, weighted and unweighted, with labelled leaves or all nodes or without labelling. The tree topology is presented using parentheses and commas: leaves are separated with comas, a set of leaves is joined into cluster by surrounding them with parentheses. The end of tree's definition is marked with a semicolon. Optionally, leaves and internal nodes are named with strings, colons after leaves and parentheses allow to mark branches lengths. Depending on the level of precision, a can be represented in Nevick format in several ways, e.i. the tree topology from figure 10.2 can be represented as in table 10.2. A formal description of the format called *"Newick's 8:45" Tree Format Standard* was produced by Gary Olsen and is available at [47] (e.i. in). Other description is available at the PHYLIP package site [83] and at Wikipedia [82].

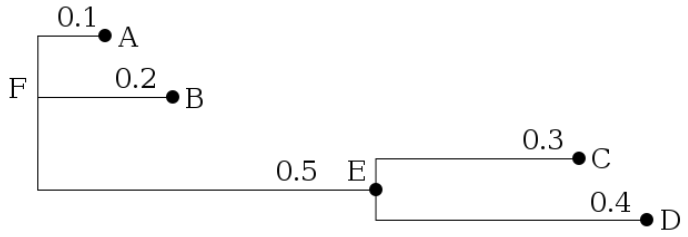


Figure 10.2: A sample figure for table tab:newick. Source: [82].

(„(,));	no nodes are named
(A,B,(C,D));	leaf nodes are named
((C,D),A,B);	the sequence of internal nodes does not matter
(A,B,(C,D)E)F;	all nodes are named
(:0.1,:0.2,(:0.3,:0.4):0.5);	all but root node have a distance to parent
(:0.1,:0.2,(:0.3,:0.4):0.5):0.0;	all have a distance to parent
(A:0.1,B:0.2,(C:0.3,D:0.4):0.5);	distances and leaf names (popular)
(A:0.1,B:0.2,(C:0.3,D:0.4)E:0.5)F;	distances and all names
((B:0.2,(C:0.3,D:0.4)E:0.5)F:0.1)A;	a tree rooted on a leaf node (rare)

Table 10.2: Different ways of Newick representation of a tree in figure 10.2. Source: modified [82].

The Bio++ tree container - a *bpp::TreeTemplate* class - is a structure which hierarchy and relations is in the opinion of PhylotreeDist creators well planned and well described in the Bio++ documentation.

In the figure 10.3 there is presented a *bpp::TreeTemplate* inheritance diagram and the figure 10.4 shows the container's collaboration with other classes. The *bpp::Tree* interface that is implemented by *bpp::TreeTemplate* provides general methods to store and manipulate phylogenetic trees. The *bpp::TreeTemplate* is the only implementation of the *bpp::Tree* interface. It uses a recursive storage of nodes - classes implementing an *bpp:N* interface which implementation is used in the PhylotreeDist the *bpp:Node* class. The *bpp::Node* contains methods to access the father and son nodes in the hierarchy, and several fields like name, id or length of the branch connected to its father. The PhylLib stores a tree by holding a reference to its root node. If a tree is unrooted - a pseudo root node is arbitrary chosen from internal nodes. Moreover, the *bpp::TreeTools* and *bpp::TreeTemplateTools* classes serve utilitary static methods. Trees can be read and written from/to files, using the *bpp::Newick* class.

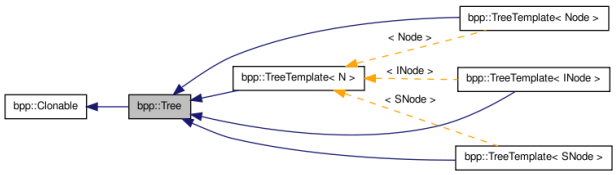


Figure 10.3: The bpp::TreeTemplate<N> inheritance diagram. [37]

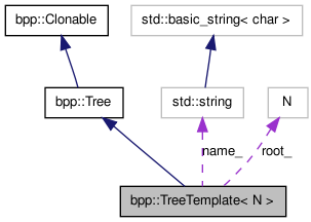


Figure 10.4: The bpp::TreeTemplate<N> collaboration diagram. [37]

10.4 Some unpleasant conclusions after using Bio++

After finishing the first release of PhylotreeDist, having some experiences in using the Bio++ PhylLib, we here present our observations and conclusions about the library:

1. Although the way the tree is stored does not depend on whether it is rooted or not, the library does not store any flag with information on whether the Newick input tree was rooted. The Bio++ documentation assumes as follows: "To deal with non-rooted trees, we place an artificial root at a particular node: hence the root node appears to be trifurcated. This is the way unrooted trees are described in the parenthetic description, the so called Newick format." [38] As a conseqience, the *bpp::TreeTemplate::isRooted()* method returns true only when the root or pseudo-root node has exactly two sons. This in fact is compatible with the original description of Newick format from [47], but it is obviously true only for the binary trees, whereas the Bio++ library documentation [38] claims that it services multifurcating trees.

2. Although in the Bio++ documentation it states, that there is driven attention to provide good time efficiency, there are several methods at least for simple trees operations, that prove to the opposite. The

following ones are the ones that fore PylotreeDist are crucial and often called:

- The $bpp :: TreeTemplate :: getNumberOfLeaves()$ method works in $O(n)$ time, scanning all the tree and counting the nodes that do not have any descendant. We claim that this could easily be implemented to work generally in $O(1)$ time, using a field that holds the number of leaves.
- The $bpp :: TreeTemplate :: isRoot(int\ id)$ method works in $O(n)$ time, scanning the tree for the node with given id. If the node found is the same as the one to which the reference in the tree object is hold, than there is returned a true value. We claim that the method could work in $O(1)$ time, checking if the stored root node has the same id as the given parameter.

3. Parsing data from Newick to $bpp :: TreeTemplate$ object is very time expensive. Comparing to e.i. pal Java package [64] and open source code from quartet distance implementation from [15] the process takes statistically over four times more time, see tables 10.3 and 10.4.

Input		Time[s]			Proportion	
Trees num.	Leaves num.	Bio++	pal	QuartetDist	Bio++ : pal	Bio++ QuartetDist
100000	43	78.077	15.256	1.959	5.2 : 1	6.61 : 1
101183	19	29.778	8.015	7.958	3.71 : 1	3,74 : 1
200	1250	3.936	1.549	1.329	2.54 : 1	2.96 : 1
200	5000	18.549	3.09	3.08	6.0 : 1	6.1 : 1
Average					4.36 : 1	4.87 : 1

Table 10.3: Comparing time results of parsing a tree from Newick input file to a trees object container for Bio++ and two java libraries: pal and QuartetDist.

Input		Time[s]		Proportion
Trees num.	Leaves num.	Trees read	Distance computation	read : computation
100000	43	78.077	15.256	5.1
101183	19	29.778	8.015	3.7
200	1250	3.936	1.549	2.5
200	5000	18.549	3.09	6
Average				4.3

Table 10.4: Proportions of time that it takes for the Bio++ to parse the trees comparing to matching split metric $O(n^3)$ computation time.

Chapter 11

PhylotreeDist

In this chapter we present the *PhylotreeDist* library which id the implementation part for this paper. We outline the functionalities, give references to the code documentation and in the next chapter - provide a manual for the library, where we also present the interface of a sample application *phylotreeDistApp* that uses the library.

11.1 Functionalities and characteristics

The *PhylotreeDist* library is a set of metrics between phylogenetic trees. It gathers currently (year 2011) best known time complexity solutions for the most popular metrics and provides a set of metrics that base on new approach: minimum weight perfect matching metrics.

The idea of writing the *PhylotreeDist* library was to provide a fast bioinformatic tool that

- implements in a time-optimal way metrics from the minimum weight perfect matching family of metrics
- gathers the most popular metrics which compare phylogenetic trees, driving special attention to implement the best known time complexity algorithms.

We also provide a sample application that makes use of the library and can the guide to its usage. (The application is described in the manual in chapter 12.)

We want this software to be free of charge and popularized. Also, to popularize it we decided to make it compatible with a Bio++ bioinformatic library (see chapter 10) and after more tests, propose the solutions to be integrated with Bio++.

In the table 11.1 we present a list of the implemented algorithms with their time and space complexity and type of trees they handle. All the algorithms implemented in the library are discussed in chapters 7 and 8.

metric	target trees*	complexity		references
		time	space	
Robinson-Foulds	no constraints	$O(n)$	$O(n)$	section 7.2 , [19]
weighted Robinson-Foulds	unrooted branch-weighted	$O(n)$	$O(n)$	section 7.2 , [19]
Quartets	unrooted	see **	$O(n^2)$	section 7.3 , [15]
Triplets	rooted	$O(n^2)$	$O(n^2)$	section 7.4 , [18]
Nodal (manhattan)	unrooted	$O(n^2)$	$O(n)$	section 7.5 , [4]
weighted Nodal (manhattan)	unrooted branch-weighted	$O(n^2)$	$O(n)$	section 7.5 , [4]
Nodal (pythagorean)	unrooted	$O(n^2)$	$O(n)$	section 7.5 , [4]
weighted Nodal (pythagorean)	unrooted branch-weighted	$O(n^2)$	$O(n)$	section 7.5 , [4]
MWPM splits	unrooted	$O(n^3)$	$O(n^2)$	section 8.2 , [7]
MWPM clusters	rooted	$O(n^3)$	$O(n^2)$	section 8.3 , [7]
MWPM pairs	rooted binary	$O(n^3)$	$O(n^2)$	section 8.3 , [7]

Table 11.1: Algorithms implemented in the PhylotreeDist library.
*The pair of the target trees must have the same leaves set.
** $O(|L| + |V_1 \setminus L|\ |V_2 \setminus L|\ min\{\Delta(T_1 \setminus L), \Delta(T_2 \setminus L)\})$.

11.2 Interface

The interface to the library is a *dist* :: *PhylotreeDist* class with only static methods that computes distances between trees passed as constant parameters. In the table 11.2 we present the names of methods in the *dist::PhylotreeDist* class for all the implemented metrics.

Each method calculates the distance between two constant trees passed as parameters, which are the *bpp::TreeTemplate<Node>* object from the Bio++ library. The user can decide through a flag passed as an argument whether

metric	method name
Robinson-Foulds	robinsonFoulds
weighted Robinson-Foulds	robinsonFouldsW
Quartet	quartetDistance
Triplets	tripletsDistance
Nodal (manhattan)	nodalDistance
weighted Nodal (manhattan)	nodalDistanceW
Nodal (pythagorean)	nodalDistance_pythagorean
weighted Nodal (pythagorean)	nodalDistanceW_pythagorean
MWPM splits	perfectMatching_splits
MWPM clusters	perfectMatching_clusters
MWPM pairs	perfectMatching_pairs

Table 11.2: Names of methods implemented in the PhylotreeDist library.

to check if the trees meet metric's constraints -this functionality is optional as it is time-consuming. Below we present a description of each method:

static int dist::PhylotreeDist::robinsonFoulds (
const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setNodesId = true,
bool checkNames = false
) throw (bpp::Exception)

The RobinsonFoulds distance between two unrooted or rooted, multifurcating trees with the same set of leaves. The RobinsonFoulds metric bases on counting occurrences of:

- for unrooted trees: bipartitions not common to both trees
- for rooted trees: clusters not common to both trees

Time complexity: $O(n)$, algorithm adapted from [19].
Parameters:

[**in**] **tr1** First tree.

[**in**] **tr2** Second tree.

[**in**] **setNodesId** *optional*. Set *true* if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$ and the internal nodes ids as a sequence $n, n+1, n+2, ...$ Defaults to *true*.

[**in**] **checkNames** *optional*. Set *true* if to check whether trees have the same leaves set. Defaults to $false$.

Returns the RobinsonFoulds distance between trees.
Exceptions: bpp::Exception if trees have different leaves sets or if both trees are not either rooted or unrooted.

static int dist::PhylotreeDist::robinsonFouldsW (
const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setNodesId = true,
bool checkNames = false
) throw (bpp::Exception)

The weighted RobinsonFoulds distance between two unrooted, branch-weighted trees with the same set of leaves. The distance is given by a following expression:

$$d_{wRF}(T_1, T_2) = \sum_{b \in \beta'(T_1) \setminus \beta'(T_2)} w_1(b) + \sum_{b \in \beta'(T_2) \setminus \beta'(T_1)} w_2(b) + \sum_{b \in \beta'(T_1) \cap \beta'(T_2)} |w_1(b) - w_2(b)|$$

where $\beta'(T_i)$ is an $|E|$-sized set of all the bipartitions of a tree T_i and $w_i(b)$ is weight of an edge of T_i which removal leads to a bipartition b. Time complexity: $O(n)$, algorithm adapted from [19].
Parameters:

[**in**] **tr1** First unrooted, branch-weighted[1]

[**in**] **tr2** Second unrooted, branch-weighted tree.

[**in**] **setNodesId** *optional*. Set *true* if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$ and the internal nodes ids as a sequence $n, n+1, n+2, ...$ Defaults to *true*.

[1] Weights on a branch are values stored by nodes of the *bpp::TreeTemplate<bpp::Node>* object as a distance to node's father (every node despite a root has got a father; if a tree is unrooted, the root is a pseudo root node, for details see 10.3). The weight value is returned through a *bpp::Node::getDistanceToFather()* method. The weights may be assigned while creating the tree object from the Newick frmat (see section 10.3) or by *bpp::Node::setDistanceToFather()* method that sets weight on branch between a node and its father.

[**in**] **checkNames** *optional*. Set *true* if to check whether trees have the same leaves set. Defaults to $false$.

Returns the weighted RobinsonFoulds distance between trees.
Exceptions: bpp::Exception if trees have different leaves sets or any tree is not branch-weighted unrooted.

static int dist::PhylotreeDist::quartetDistance (
const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setLeavesId = true,
bool checkNames = false
) throw (bpp::Exception)

The Quartet distance between two unrooted trees with the same set of leaves. The method counts the number of sets of four species for which the topologies differ in the two trees.
Time complexity: $O(|L| + |V_1 \setminus L|\ |V_2 \setminus L|\ min\{\Delta(T_1 \setminus L), \Delta(T_2 \setminus L)\})$.
Algorithm adapted from [15]

Parameters:

[**in**] **tr1** First unrooted tree.

[**in**] **tr2** Second unrooted tree.

[**in**] **setLeavesId** *optional*. Set *true* if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$. Defaults to *true*.

[**in**] **checkNames** *optional*. Set *true* if to check whether trees have the same leaves set. Defaults to $false$.

Returns the quartet distance between trees.
Exceptions: bpp::Exception if trees have different leaves sets or any tree is not unrooted.

static int dist::PhylotreeDist::tripletsDistance (
const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setLeavesId = true,
bool checkNames = false

) throw (bpp::Exception)

The Triplets distance between two rooted bifurcating trees with the same set of leaves. The method counts the number of subtrees of three taxa that are different in the trees. The distance is called in [18] a close cousin of the quartet metric for unrooted trees.
Time complexity: $O(n^2)$. Algorithm adapted from [3]

Parameters:

[**in**] **tr1** First rooted tree.

[**in**] **tr2** Second rooted tree.

[**in**] **setLeavesId** *optional*. Set *true* if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$. Defaults to *true*.

[**in**] **checkNames** *optional*. Set *true* if to check whether trees have the same leaves set. Defaults to $false$.

Returns the triplets distance between trees.
Exceptions: bpp::Exception if trees have different leaves sets or any tree is not rooted.

static int dist::PhylotreeDist::nodalDistance (
const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setLeavesId = true,
bool checkNames = false
) throw (bpp::Exception)

The Nodal distance with use of manhattan metric between two unrooted trees with the same set of leaves.
For each pair of leaves in each tree there is counted a number of branches on the path between the two leaves. Next for each pair of leaves this value is compared between T_1 and T_2 with use of manhattan metric.
Time complexity: $O(n^2)$. Algorithm adapted from [4].

Parameters:

[**in**] **tr1** First unrooted tree.

[**in**] **tr2** Second unrooted tree.

[**in**] **setLeavesId** *optional*. Set *true* if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$. Defaults to *true*.

[**in**] **checkNames** *optional*. Set *true* if to check whether trees have the same leaves set. Defaults to $false$.

Returns the nodal manhattan distance between trees.
Exceptions: bpp::Exception if trees have different leaves sets or any tree is rooted.

static int dist::PhylotreeDist::nodalDistanceW (
const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setLeavesId = true,
bool checkNames = false
) throw (bpp::Exception)

The weighted nodal distance with use of manhattan metric between two unrooted branch-weighted trees with the same set of leaves.
For each pair of leaves in each tree there is computed a sum of weights on branches that build a path between the pair of leaves. Next for each pair of leaves their distance in trees T_1 and T_2 are compared with use of manhattan metric.
Time complexity: $O(n^2)$. Algorithm adapted from [4].

Parameters:

[**in**] **tr1** First unrooted branch-weighted tree.

[**in**] **tr2** Second unrooted branch-weighted tree.

[**in**] **setLeavesId** *optional*. Set *true* if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$. Defaults to *true*.

[**in**] **checkNames** *optional*. Set *true* if to check whether trees have the same leaves set. Defaults to $false$.

Returns the weighted nodal manhattan distance between trees.
Exceptions: bpp::Exception if trees have different leaves sets or branches are not weighted or any tree is rooted.

static double dist::PhylotreeDist::nodalDistance_pythagorean (
const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setLeavesId = true,
bool checkNames = false
) throw (bpp::Exception)

The Nodal distance with use of pythagorean metric between two unrooted trees with the same set of leaves.
For each pair of leaves in each tree there is counted a number of branches on the path between the two leaves. Next for each pair of leaves their distance in trees T_1 and T_2 are compared with use of pythagorean metric.
Time complexity: $O(n^2)$. Algorithm adapted from [4].

Parameters:

[**in**] **tr1** First unrooted tree.

[**in**] **tr2** Second unrooted tree.

[**in**] **setLeavesId** *optional*. Set *true* if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$. Defaults to *true*.

[**in**] **checkNames** *optional*. Set *true* if to check whether trees have the same leaves set. Defaults to $false$.

Returns the nodal pythagorean distance between trees.
Exceptions: bpp::Exception if trees have different leaves sets or any tree is rooted.

static double dist::PhylotreeDist::nodalDistanceW_pythagorean (

const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setLeavesId = true,
bool checkNames = false
) throw (bpp::Exception)

The Nodal distance with use of pythagorean metric between two unrooted, branch-weighted trees with the same set of leaves.
For each pair of leaves in each tree there is computed a sum of weights on branches that build a path between the pair of leaves. Next for each

pair of leaves their distance in trees T_1 and T_2 are compared with use of pythagorean metric.
Time complexity: $O(n^2)$. Algorithm basing on [4].

Parameters:

[**in**] **tr1** First unrooted branch-weighted tree.

[**in**] **tr2** Second unrooted tree.

[**in**] **setLeavesId** *optional*. Set *true* if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$. Defaults to *true*.

[**in**] **checkNames** *optional*. Set *true* if to check whether trees have the same leaves set. Defaults to $false$.

Returns the weighted nodal pythagorean distance between trees.
Exceptions: bpp::Exception if trees have different leaves sets or branches are not weighted or any tree is rooted.

static int dist::PhylotreeDist::perfectMatching_splits (
const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setLeavesId = true,
bool checkNames = false
) throw (bpp::Exception)

The Minimum Weight Perfect Matching distance between two unrooted trees with the same set of leaves with splits as description elements.
The algorithm performs a minimum weight perfect matching on a weighted bipartite graph, where a graph's partition corresponds to a description elements set of a tree - a set of all the tree's splits (bipartitions). The weight on a graph's edge that links two splits $A_1|B_1$ and $A_2|B_2$ is calculated as $h(A_1|B_1, A_2|B_2) = min(|A_1 \oplus A_2|, |A_1 \oplus B_2|) = min(|A_1| + |A_2| - 2|A_1 \cap A_2|, n - (|A_1| + |A_2| - 2|A_1 \cap A_2|)$.
Time complexity: $O(n^3)$. Algorithm adapted from [7].
By default the minimum weight perfect matching is computed by an external library [79] that bases on Jonker and Volgenant algorithm. If during compilation there is a preprocessor HUNGARIAN symbol defined (-DHUNGARIAN), then there is a Knuth's Hungarian algorithm used, adapted from [48].

Parameters:

[**in**] **tr1** First unrooted tree.

[**in**] **tr2** Second unrooted tree.

[**in**] **setLeavesId** *optional*. Set *true* if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$. Defaults to *true*.

[**in**] **checkNames** *optional*. Set *true* if to check whether trees have the same leaves set. Defaults to $false$.

Returns the splits minimum weight perfect matching distance between trees.
Exceptions: bpp::Exception if trees have different leaves sets or any tree is not unrooted.

static int dist::PhylotreeDist::perfectMatching_clusters (
const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setLeavesId = true,
bool checkNames = false
) throw (bpp::Exception)

The Minimum Weight Perfect Matching distance between two rooted trees with the same set of leaves with clusters as description elements. The algorithm performs a minimum weight perfect matching on a weighted bipartite graph, where a graph's partition corresponds to a description elements set of a tree - a set of all the tree's clusters. The weight on a graph's edge that links two clusters A_1 and A_2 is calculated as $h(A_1, A_2) = |A_1 \oplus A_2| = |A_1| + |A_2| - 2|A_1 \cap A_2|$.
Time complexity: $O(n^3)$Algorithm main assumptions base on [7].
By default the minimum weight perfect matching is computed by an external library [79] that bases on Jonker and Volgenant algorithm. If during compilation there is a preprocessor HUNGARIAN symbol defined (-DHUNGARIAN), then there is a Knuth's Hungarian algorithm used, adapted from [48].

Parameters:

[**in**] **tr1** First rooted tree.

[**in**] **tr2** Second rooted tree.

[**in**] **setLeavesId** *optional*. Set *true* if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$. Defaults to *true*.

[**in**] **checkNames** *optional*. Set *true* if to check whether trees have the same leaves set. Defaults to $false$.

Returns the clusters minimum weight perfect matching distance between trees.
Exceptions: bpp::Exception if trees have different leaves sets or any tree is not rooted.

static int dist::PhylotreeDist::perfectMatching_pairs (
const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setLeavesId = true,
bool checkNames = false
) throw (bpp::Exception)

The Minimum Weight Perfect Matching distance between two rooted binary trees with the same set of leaves with a set of pairs as a description element. The algorithm performs a minimum weight perfect matching on a weighted bipartite graph, where a graph's partition corresponds to a description elements set of a tree - a set of all two-element sets for a particular internal node, where each two-element set corresponds to the pair of leaves that have a particular node as the most recent common ancestor. The weight on a graph's edge that links two sets of pairs A_1 and A_2 is calculated as $h(A_1, A_2) = |A_1 \oplus A_2| = |A_1| + |A_2| - 2|A_1 \cap A_2|$.
Time complexity: $O(n^3)$. Algorithm main assumptions base on [7].
By default the minimum weight perfect matching is computed by an external library [79] that bases on Jonker and Volgenant algorithm. If during compilation there is a preprocessor HUNGARIAN symbol defined (-DHUNGARIAN), then there is a Knuth's Hungarian algorithm used, adapted from [48].

Parameters:

[**in**] **tr1** First binary rooted tree.

[**in**] **tr2** Second binary rooted tree.

[**in**] **setLeavesId** *optional*. Set *true* if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$. Defaults to *true*.

[**in**] **checkNames** *optional*. Set *true* if to check whether trees have the same leaves set. Defaults to $false$.

Returns the pairs minimum weight perfect matching distance between trees.
Exceptions: bpp::Exception if trees have different leaves sets or any tree is not binary rooted.

11.3 Implementation

The interface to the library is a *dist* :: *PhylotreeDist* class with only static methods. Each distance has its own method which generally follows or have similar the signature to
static int dist::PhylotreeDist::trees_metric_name (
const bpp::TreeTemplate< bpp::Node > & trIn1,
const bpp::TreeTemplate< bpp::Node > & trIn2,
bool setNodesId = true,
bool checkNames = false
) throw (bpp::Exception)
where the parameters are:
[in] **tr1** First input tree.
[in] **tr2** Second input tree.
[in] **setLeavesId** Set true if the two trees do not have the same ids for the same leaves or the ids are not numbered $0..n-1$.
[in] **checkNames** Set true if user wants to check trees constraints which depend on the metric, e.i. whether trees have the same leaves set, whether they are rooted or bifurcating.

Most of the metrics algorithms base on arrays that use leaf id number (id) as an indicator of a position under which data about the leaf are stored. As in *bpp::TreeTemplate<bpp::N>* objects the way to refer to the same leaves in different trees is by nodes names and as the ids of the nodes are arbitrary given, there is a need to map the leaves names to ids in trees. The solution used in PhylotreeDist is to change input trees ids so that for each of the two input trees, a taxon have the same id. Moreover the leaves ids are the numbers $[0..|L|-1]$ and internal nodes have ids $|L|, |L|+1, |L|+2, ...$, which is suitable for e.i. the Robinson-Foulds PhylotreeDist implementation. Moreover, if one tree is used multiple of times, then its ids can be modified once and passed to the metric with the common for all the metrics *bool setLeavesId* parameter set to true.

The function that sets the ids to be common for the input trees is *static*

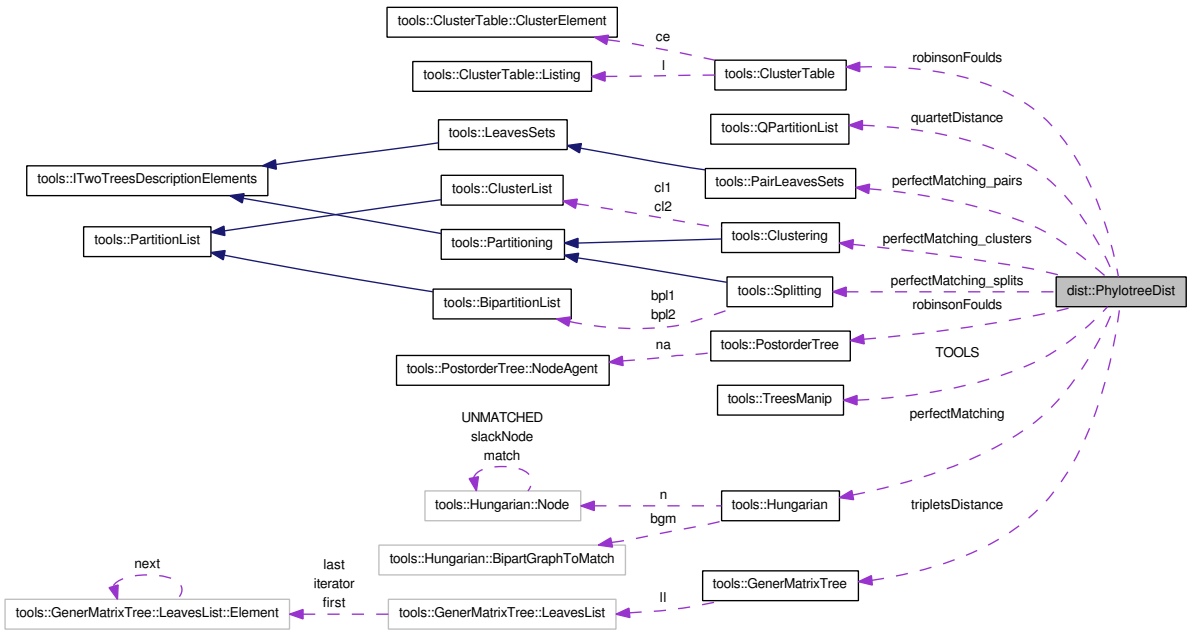


Figure 11.1: The PhylotreeDist class collaboration diagram.

TreeTemplate<Node> TreesManip::createOrderedTrees(const TreeTemplate<Node>& trIn)*. It clones an input tree, orders its leaves alphabetically, sets their ids according to this order and sets the internal nodes ids with numbers $|L|, |L|+1, |L|+2, ...$ The method returns the cloned tree. The time complexity of the method is $O(n \log n)$

The library consists of two namespaces: *dist::* and *tools::*. The *dist::* holds the interface to the library; refers to the only one class *PhylotreeDist*. The *tools::* joins the helper classes with structures and methods which are key elements to get the distances. The class collaboration diagram is shown on figure 11.1. The *dist::PhylotreeDist* is connected to classes that are helper while computing distances: for the containers that are used for a particular metric, the label on a connection arrow indicates the metric's name. The containers point to their ancestors in class hierarchy (solid arrows) and objects they make use of (dashed arrows). There is also a *tools::TreesManip* static methods class that provides different tools shared by the library's classes.

The general pattern of each method proceedings is shown on the activity diagram in the figure 11.2. The information for decision nodes is passed as method's parameters. After the optional initial operations, there are created structures that store description elements. Finally, for each description element there is calculated a distance to the tree it does not belong to and total

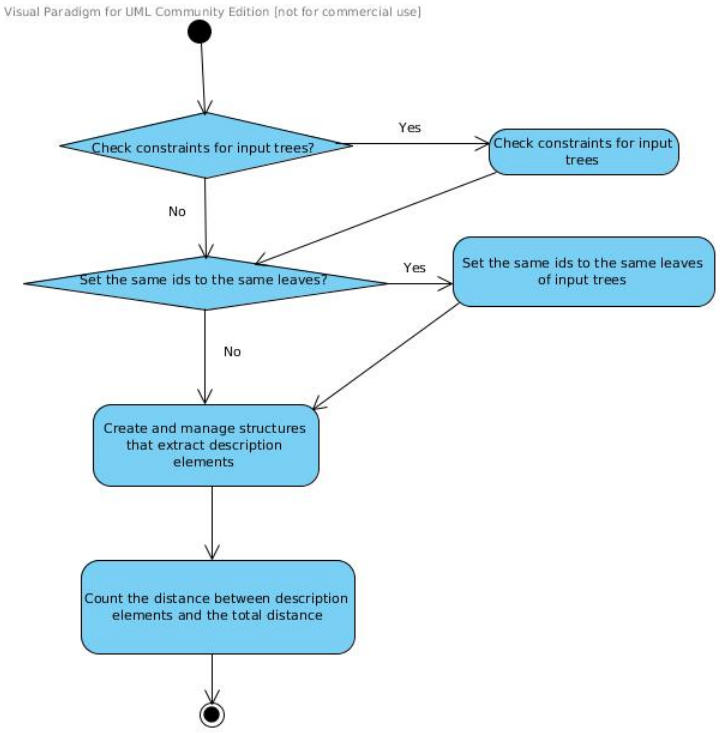


Figure 11.2: The general pattern of proceedings for each *dist::PhylotreeDist* method.

distance is computed, which is the returned value.

11.3.1 MWPM implementation details

Calculating a minimum weight perfect matching distance consists of the following operations: extracting description elements, computing distance matrix of distances between each element of one tree with each element of other tree and finally resolving the assignment problem for the matrix, returning the weight of the matching. All the MWPM algorithms implemented in the PhylotreeDist library follow the following procedure:

There is created an object that inherits the $tools :: ITwoTreesDescriptionElements$ interface. The object is a container that stores the two input trees' description elements and distances among the elements. Extracting description elements from both trees takes place in the container's constructor. The container also must implement the interface functions:

- *int tools::ITwoTreesDescriptionElements::getCostMatrix(int** costMatrix)* - computes a cost matrix between each pair of description elements. The costs are returned in the $costMatrix$ parameter which is an n-elements array of n-elements arrays of integer values - therefore it is a square $n \times n$ matrix. The n value is obtained from by the *tools::ITwoTreesDescriptionElements::getSize()* method. The $costMatrix$ needs to be initialized before passing to the function. The method returns the size of matrix dimension.
- *int tools::ITwoTreesDescriptionElements::getSize()* - returns the number of description elements.

The container is passed as the first parameter to the *static int dist::PhylotreeDist::getPMDistance(tools::ITwoTreesDescriptionElements& descriptionElements)* method. The method returns the weight of the minimum weight perfect matching for the cost matrix from the $tools :: ITwoTreesDescriptionElements :: getCostMatrix$ method.

Figure 11.3 is an inheritance diagram of the three description elements containers of the three implemented MWPM distances: MWPM splits, MWPM clusters and MWPM pairs.

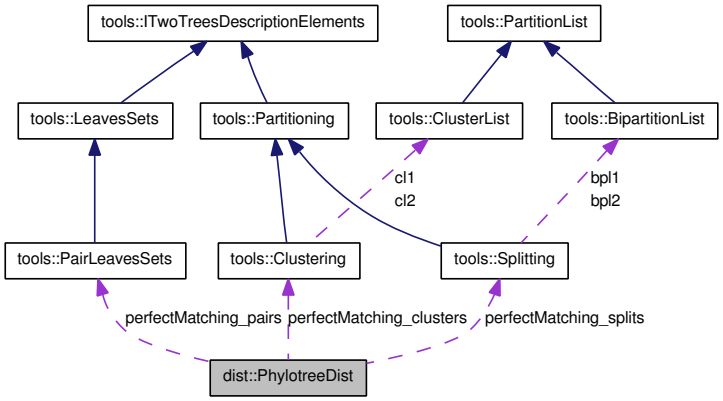


Figure 11.3: The general pattern of proceedings for each *dist::PhylotreeDist* method.

11.3.2 Performance

We attempted to make the *PhylotreeDist* a time-efficient library. In consequence, we chose the C++ language, took special care about complexity in choosing algorithms and about code implementation, conducted performance tests that leaded to code optimizations.

Using the valgrind [92] tool, we located the most inefficient elements of PhylotreeDist. This concerned mainly the performance of Bio++ tree container implementation and tree methods. With figures 11.4 and 11.5 produced by the valgrind tool, we present an instance of encountered problems. Both figures shows graphs with the percentage of the time spent on particular operations of function that returns the nodal distance in time that we assumed to be $O(n^2)$ (input: 3 phylogenetic trees of 250 leaves; an execution performs two comparisons: one between 1^{st} and 2^{nd} input trees and one between 2^{nd} and 3^{rd} input trees).

In figure 11.4 over 97% of time used the method that counts the distance is spent on checking whether the node with the id given as a parameter is a root. This is the Bio++ method *bpp::TreeTemplate<bpp::Node>::isRoot(int)* that we call $O(n^2)$ times assuming that it performs in $O(1)$ time. However, in the Bio++ implementation, the method traverses all the tree, searching for a node with given id. To make matters worse, it does not stop when the node is encountered, but checks if somehow there are more nodes with that id (which by assumption should never take place).

After implementing our own function that checks if a node is root that sim-

ply compares the requested id to root's id, the percentage of time spent on computing the distance was 7% of all the running time in contrast to previous 79% (see figure 11.5). This fact leads to another conclusion: in this case over 89% of time spent on the execution of the application was spent on Bio++ method responsible for parsing the newick file input of only three 250-leaves trees to the Bio++ tree container *bpp::TreeTemplate<Node>* (this is discussed more detailed in the 10.4 section). This is more controversial considering the fact that the nodal distance requires as much as $O(n^2)$ time. The Bio++ code responsible for parsing from the Newick format is rather illegible and hard to understand. As we want our library to be compatible with Bio++, we did not change trees representation. However we would very welcome if Bio++ would consider some of our suggestions that we are going to send.

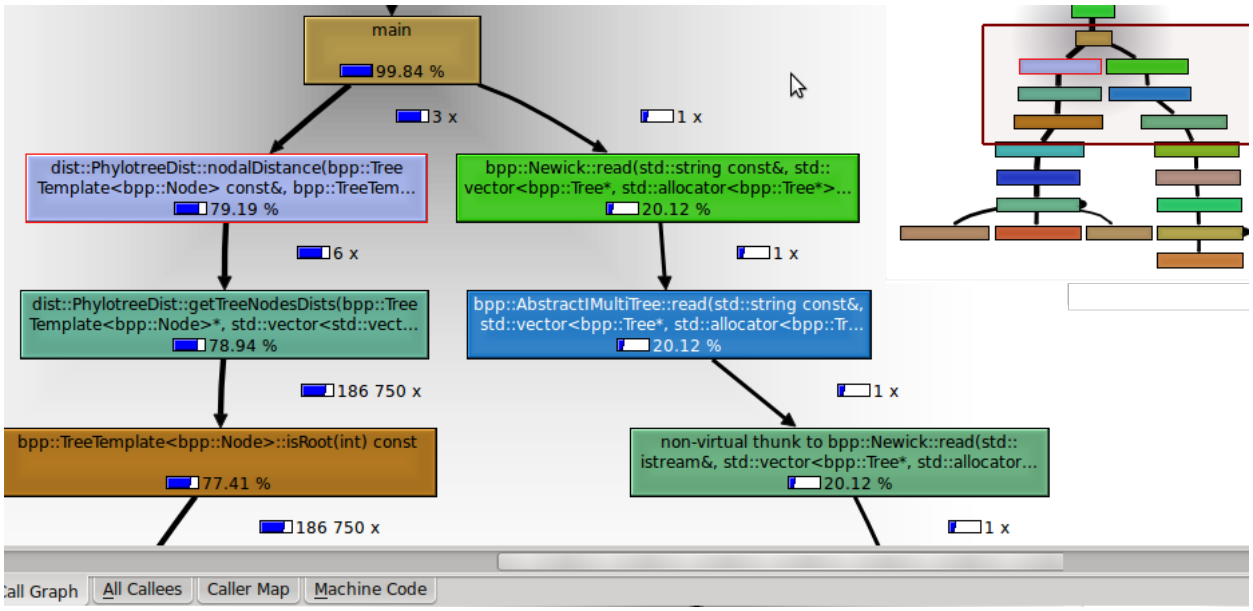


Figure 11.4: Valgrind output for the method that parses Newick input and coputes nodal distance with use of Bio++ *isRoot*(*int*) method.

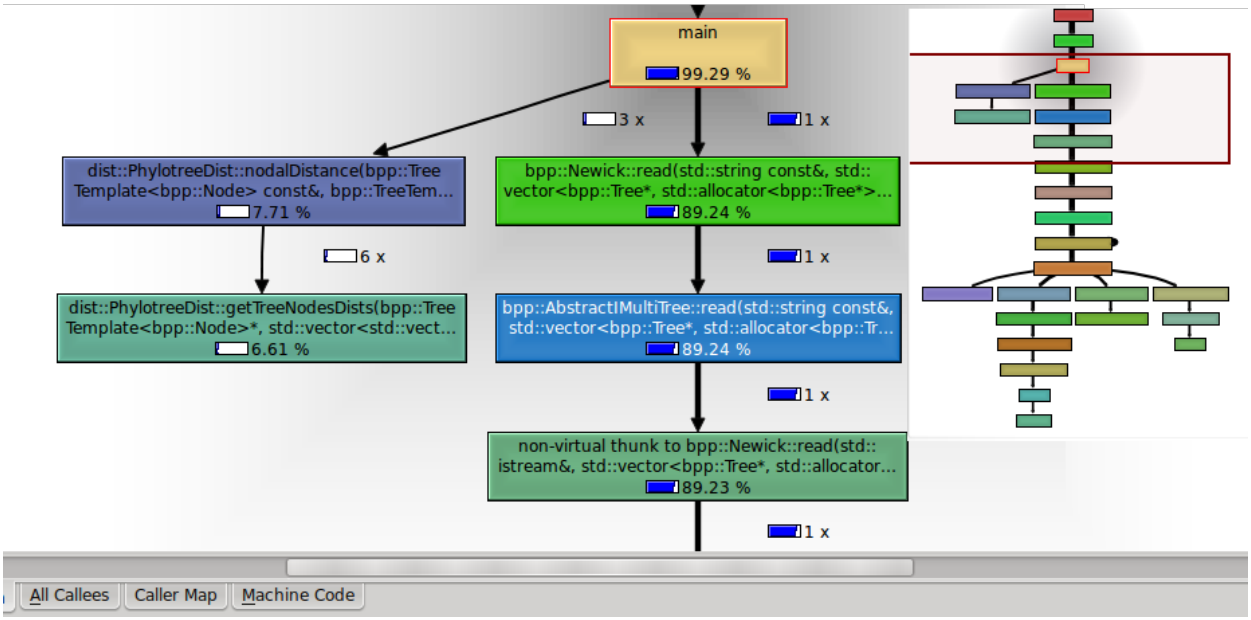


Figure 11.5: Valgrind output for the method that parses Newick input and coputes nodal distance without use of Bio++ $isRoot(int)$ method.

Chapter 12

PhylotreeDist manual

PhylotreeDist is a C++ bioinformatic library that provides various phylogenetic distances between pair of trees.

PhylotreeDist is distributed with source code as well as binaries for Linux (x86) environment.

Available distances:

- **Perfect matching - on splits** for unrooted trees
- **Perfect matching - on clusters** for rooted trees
- **Perfect matching - on pairs** for rooted binary trees
- **Robinson-Foulds** for rooted and unrooted trees
- **Weighted Robinson-Foulds** for unrooted branch-weighted trees
- **Quartet** for unrooted trees
- **Triplets** for rooted trees
- **Nodal** for unrooted trees
- **Weighted Nodal** for unrooted branch-weighted trees

12.1 Dependences

Bio++ [40]
For required resources, visit `http://biopp.univ-montp2.fr/articles/download/` and download the packages of **exact version** as given below (as currently in July 2011 the newest version is invalid). The recommended sequence of installation is the order of the listing below.

1. bpp-core 2.0.0
2. bpp-utils 1.5.0
3. bpp-numcalc 1.8.0
4. bpp-seq 1.7.0
5. bpp-phyl 1.9.0

You can download the required libraries via `http://www.awek.com.pl/~andziaania/libbpp.tar.gz`
place the library files into `externals/lib` directory and add the `externals/lib` to the *LD_LIBRARY_PATH* variable (see section 12.3.1).

12.2 Resources

The library with sample application are available via Internet:

- Linux(x86) binaries
  `http://www.awek.com.pl/~andziaania/PhylotreeDist_LBin.tar.gz`
- Source code with binaries for Linux(x86)
  `http://www.awek.com.pl/~andziaania/PhylotreeDist_src.tar.gz`
  The compilation process is described in section 12.4.2

You can unpack the resources e.i. using tar [90] archiving utility

```
tar -xvvzf PhylotreeDist_XXX.tar.gz
```

12.3 Usage

The PhylotreeDist package can be used as a **library** or a **commandline executable**.

12.3.1 Commandline executable

1. Download Bio++ dependences (section 12.1).
   if needed - set LD_LIBRARY_PATH to them:
   ```
   export LD_LIBRARY_PATH=\
   $LD_LIBRARY_PATH:<path to Bio++ libraries>
   ```

2. Download and manage appropriate PhylotreeDist resources (see section 12.2).

3. Enter into the PhylotreeDist directory with binaries.

4. Show the library directory to the system:
   ```
   export LD_LIBRARY_PATH=\
   $LD_LIBRARY_PATH:<path to phylotreedist library e.i. current dir>
   ```

5. The executable is run according to the following pattern:
   ```
   ./phylotreeDistApp -i <inputFile.newick> -o <outputFile> \
   [ -d <distance type> ] [ -m <comparison mode> ] [-c]
   ```
   phylotreeDist executable arguments:

   -i <*inputFile.newick*>
   Compulsory.
   Indicates the source file with trees in newick format.

   -o <*outputFile*>
   Compulsory.
   Indicates the output file.

   -m [p|m]
   Default to p.
   Optional.
   Comparison mode - which input trees are to be compared together.
   Options:
   p - pair - every two neighboring trees in the input file are compared
   m - matrix - every two trees in the input file are compared

   -d [ms|mc|mp|rf|rfw|q|t|n|nw|nm|nmw|np|npw]
   Default to rf.
   Optional.
   Distance choice - the algorithm used to compare the trees.
   Options:

ms	unrooted trees	matching with splits as description elements
mc	rooted trees	matching with clusters as description elements
mp	rooted binary trees	matching with pairs as description elements
rf	two rooted or two unrooted trees	robinson-foulds (default if no metric is chosen)
rfw	unrooted branch-weighted trees	weighted robinson-foulds
q	unrooted trees	quartets
t	rooted trees	triplets
nm *or* n	unrooted trees	nodal, computing distance between leaves with manhattan metric
nmw *or* nw	unrooted branch-weighted trees	weighted nodal, computing distance between leaves with manhattan metric
np	unrooted trees	nodal, computing distance between leaves with pythagorean metric
npw	unrooted branch-weighted trees	weighted nodal, computing distance between leaves with pythagorean metric

-c

Optional.
Check trees constraints (un/rooted, bi/multifurcating, the same leaves sets) and throw exception if anything is incorrect.

RESULTS

The distances results are available in the file given as the *-o* parameter value. The results are in the following format:

Metric:	*metric name*
Comparison mode:	*comparison mode type*
Input file:	*filename*
Output file:	*filename*
———	
comparison id (1..n)	*<tabulation separator>* *distance value*

Example:

Metric:	Matching-Splits
Comparison mode:	pairs
Input file:	data/unrooted.newick
Output file:	msResult.dat
———	
1	137
2	25
3	4
4	144

12.3.2 Library

The interface to the library is provided in the file
`include\PhylotreeDist.h`
Also you can consult chapter 11

The dynamic library is available as `libPhylotreeDist.so`.

12.4 Source code management

12.4.1 Directory structure

`bin\`		binaries
`bin\`	`GNU-Linux-x86\`	Linux(x86) `phylotreeDistApp` and `libPhylotreeDist.so` binaries
`data\`		sample trees in Newick format
`doc\`		documentation

`externals\`	header files of Bio++
`include\`	header files of the PhylotreeDist library
`Makefile`	makefile
`makeDepend\`	dependences for the Makefile
`sampleapp\`	sample application `PhylotreeDistApp` source code and Makefile
`src\`	source files of the PhylotreeDist library
`test\`	unit test files for the PhylotreeDist library

12.4.2 Compilation

Enter into `PhylotreeDist` directory.
To compile the library `libPhylotreeDist.so`:

```
make
```

To compile the executable sample application `PhylotreeDistApp`
first compile the library and then:

```
cd PhylotreeDistApp
make
cd ..
```

The binaries (dynamic `libPhylotreeDist` library and executable `PhylotreeDistApp` sample application) are available at `bin\`<*platform*>.

12.5 Licence

Summary

Beksiński, 1975, [98]

All progress is precarious,
and the solution of one problem brings us face to face with another problem.

Martin Luther King, Jr.

Chapter 13

Summary and Results

This thesis results in two components: the PhylotreeDist library and this paper. The **PhylotreeDist** library is a package that gathers which compute distances among phylogenetic trees ensuring their good time performance. Moreover, the library is the second (after the D. Bogdanowicz's Java version [42]) implementation of the metrics of the minimum weight perfect matching (MWPM) family and the first implementation of the pairs and clusters MWPM instances.

This paper, **The Methods of Phylogenetic Trees Metrics Definition**, is a comprehensive background to PhylotreeDist library and provides information on different domains related to the library. The document joins the PhylotreeDist documentation, the attempt to systematize proceedings of defining phylogenetic trees distance methods, the basis and application of phylogenesis and the discussion about libraries related to PhylotreeDist.

In total, the four major products of this thesis are:

1. implementation of threes comparing MWPM metrics and their detailed theoretical analysis
2. implementation and theoretical analysis of classical phylogenetic trees metrics with emphasis put on best time performance algorithms
3. individual author's analysis and classification of phylogenetic trees comparison metrics which leads to giving their universal construction process model
4. suggestions for Bio++ library improvements

Other, related results include the analysis of:

- trees comparison metrics, their correctness of definition and their algorithms
- phylogeny appliance in the modern world
- different bioinformatic libraries written in C++ and trees distances libraries written in different programming languages
- phylogenetic package of the Bio++ library

We hope that our further attempts concerning the thesis will lead to

1. help in future researches on trees distances metrics, especially the MWPM metric, in particular at the mother Gdansk University of Technology
2. including the library to the Felsenstein/Kuhner lab bioinformatic packages list [86]
3. including the phylogenetic trees distances implementations to the Open Source Bio++ library
4. adapting to Bio++ some of the suggestion about the code

Bibliography

[1] J. Albert, J. Wahlberg, T. Leitner, D. Escanilla, and M. Uhlen. Analysis of a rape case by direct sequencing of the human immunodeficiency virus type 1 pol and gag genes. *Journal of virology*, 68(9):18–24, 1994.

[2] C. Arnold, P. Balfe, and J. P. Clewley. Sequence distances between env genes of hiv-1 from individuals infected from the same source: Implications for the investigation of possible transmission events. *Virology*, 211:198–203, 1995.

[3] M. Bansal, J. Dong, and D. Fernández-Baca. Comparing and aggregating partially resolved trees. *LATIN 2008: Theoretical Informatics*, 17:72–83, 2008.

[4] J. Bluis and D. G. Shin. Nodal distance algorithm: Calculating a phylogenetic tree comparison metric. In *Bioinformatics and Bioengineering, 2003. Proceedings. Third IEEE Symposium on*, pages 87–94. IEEE, 2003.

[5] D. Bogdanowicz. Comparing phylogenetic trees using a minimum weight perfect matching. In *Information Technology, 2008. IT 2008. 1st International Conference on*, pages 1–4. IEEE, 2008.

[6] D. Bogdanowicz. Analyzing sets of phylogenetic trees using topology metrics. *Application Mathematics*, 38:1–16, 2011.

[7] D. Bogdanowicz and K. Giaro. Comparing arbitrary unrooted phylogenetic trees using generalized matching split distance. In *Information Technology (ICIT), 2010 2nd International Conference on*, pages 259–262. IEEE, 2010.

[8] J. A. Bondy and U. Murty. *Graph Theory with Application*. London: Maxmillan Press, 1976.

[9] G. S. Brodal, R. Fagerberg, and C. N. S. Pedersen. Computing the quartet distance between evolutionary trees in time o(n log n). *Algorithmica*, 38(2):377–395, 2004.

[10] D. R. Brooks and D. A. McLennan. *Phylogeny, ecology, and behavior*. University of Chicago Press Chicago, 1991.

[11] T. A. Brown. *Genomes*. Oxford: Wiley-Liss, 2002. `http://www.ncbi.nlm.nih.gov/books/NBK21122/#A8883`.

[12] D. Bryant. Building trees, hunting for trees, and comparing trees-theory and methods in phylogenetic analysis. 1997. Unpublished PhD Thesis. Department of Mathematics, University of Canterbury, New Zealand.

[13] D. Burago, Y. Burago, and S. Ivanov. *A Course in Metric Geometry*. American Mathematical Society Providence, 2001.

[14] C. Christiansen, T. Mailund, C. Pedersen, and M. Randers. Computing the quartet distance between trees of arbitrary degree. *Algorithms in Bioinformatics*, 3692:77–88, 2005.

[15] C. Christiansen, T. Mailund, C. N. S. Pedersen, M. Randers, and M. S. Stissing. Fast calculation of the quartet distance between trees of arbitrary degrees. *Algorithms for Molecular Biology*, 1(1):1–16, 2006.

[16] J. Cohen. Bioinformatics — an introduction for computer scientists. *ACM Computing Surveys (CSUR)*, 36(2):122–158, 2004.

[17] T. H. Cormen. *Introduction to Algorithms*. The MIT press, 2001.

[18] D. E. Critchlow, D .K. Pearl, and C. Qian. The triples distance for rooted bifurcating phylogenetic trees. *Systematic biology*, 45(3):323, 1996.

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

[20] J. Dutheil, S. Gaillard, E. Bazin, S. Glémin, V. Ranwez, N. Galtier, and K. Belkhir. Bio++: a set of c++ libraries for sequence analysis, phylogenetics, molecular evolution and population genetics. *BMC Bioinformatics*, 7(1):188, 2006.

[21] M. Fourment and M. Gillings. A comparison of common programming languages used in bioinformatics. *BMC bioinformatics*, 9(1):82, 2008. `http://www.biomedcentral.com/1471-2105/9/82`.

[22] D Futuyma. *Evolutionary Biology*. Third edition. Sunderland, MA: Sinauer Associates, 1998.

[23] K. Giaro. Lecture materials for the elements of bioinformatics lecture, gdansk university of technology. Gdansk 2010.

[24] M. F. Janowitz. *Bioconsensus*. Amer Mathematical Society, 2003.

[25] M. Kubale. *Introduction to Computational Complexity and Algorithmic Graph Coloring*. Gdanskie Towarzystwo Naukowe, 1998.

[26] J. Pevsner. *Bioinformatics and functional genomics*. Wiley Online Library, 2009.

[27] E. Prugovecki. *Quantum mechanics in Hilbert space*, volume 92. Academic Pr, 1981.

[28] K. E. Robbins, P. J. Weidle, T.M. Brown, A. M. Saekhou, B. Coles, S. D. Holmberg, T. M. Folks, and M.L. Kalish. Molecular analysis in support of an investigation of a cluster of hiv-1-infected women. *AIDS research and human retroviruses*, 18(15):1157–1161, 2002.

[29] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1-2):131–147, 1981.

[30] K. A. Ross and C. R. B. Wright. Matematyka dyskretna. *PWN, Warszawa*, 1996.

[31] A. Schrijver. *Combinatorial Optimization*. Springer, 2003.

[32] M. Sobczyński, P. Mackiewicz, D. Mackiewicz, K. Smolarczyk, and S. Cebrat. Nowe spojrzenie na filogenezę z punktu widzenia genomiki. *Biotechnologia*, 3(70):102–117, 2005.

[33] K. Spalik and M. Piwczynski. Rekonstrukcja filogenezy i wnioskowanie filogenetyczne w badaniach ewolucyjnych. *KOSMOS*, 58(3-4):485–498, 2009.

[34] M. A. Steel and D. Penny. Distributions of tree comparison metrics - some new results. *Systematic Biology*, 42(2):126–141, 1993.

[35] C. S. Tsai. *Biomacromolecules: introduction to structure, function and informatics*. Wiley-Blackwell, 2007.

[36] Bio++ development forum. `http://groups.google.com/group/biopp-devel-forum`.

[37] Bio++ documentation. `http://biopp.univ-montp2.fr/articles/documentation/`.

[38] Bio++ documentation TreeTemplate class. `http://biopp.univ-montp2.fr/Documents/ClassDocumentation/bpp-phyl/html/classbpp_1_1TreeTemplate.html`.

[39] Bio++ usage forum. `http://groups.google.com/group/biopp-help-forum`.

[40] Bio++ webpage. `http://biopp.univ-montp2.fr/`.

[41] Bio++ wiki page. `http://biopp.univ-montp2.fr/wiki/index.php/Main_Page`.

[42] D. Bogdanowicz and K Giaro. *Matching Split distance for unrooted binary phylogenetic trees site*. `http://www.kaims.pl/~dambo/msdist`.

[43] CeCILL licence webpage. `http://www.cecill.info/index.en.html`.

[44] Cygwin webpage. `http://www.cygwin.com/`.

[45] Doxygen webpage. `http:/www.doxygen.org/`.

[46] Encyklopedia PWN 2010. `http://encyklopedia.pwn.pl/`.

[47] G. Olsen. The "Newick's 8:45" Tree Format. `http://evolution.genetics.washington.edu/phylip/newick_doc.html`.

[48] M. Golin. *Bipartite Matching and the Hungarian Method*. Hong Kong University of Science and Technology Course Notes `http://www.cse.ust.hk/~golin/COMP572/Notes/Matching.pdf/`.

[49] Library BIRCH. `http://home.cc.umanitoba.ca/~psgendb/`.

[50] Library BTL. `http://people.cryst.bbk.ac.uk/~classlib/bioinf/BTL99.html`.

[51] Library BTL-article. `http://bioinformatics.oxfordjournals.org/content/17/8/729`.

[52] Library Clann. http://bioinf.may.ie/software/clann/.

[53] Library COMPONENT. http://taxonomy.zoology.gla.ac.uk/rod/cplite/guide.html.

[54] Library COMPONENT distance description. http://taxonomy.zoology.gla.ac.uk/rod/cplite/ch5.pdf.

[55] Library EMBOSS. http://emboss.sourceforge.net/what/.

[56] Library EMBOSS distance description. http://emboss.bioinformatics.nl/cgi-bin/emboss/help/ftreedist.

[57] Library EMBOSS trees distances. http://emboss.bioinformatics.nl/cgi-bin/emboss/help/ftreedist.

[58] Library EMBOSS wiki. http://www.open-bio.org/wiki/Main_Page.

[59] Library libcov. http://www.biomedcentral.com/1471-2105/6/138.

[60] Library libsequence. http://www.molpopgen.org/software/libsequence.html.

[61] Library Mesquite. http://mesquiteproject.org.

[62] Library Murka. http://phylomurka.sourceforge.net.

[63] Library NCBI documentation. http://www.ncbi.nlm.nih.gov/IEB/ToolBox/CPP_DOC/.

[64] Library PAL. http://www.cebl.auckland.ac.nz/pal-project/.

[65] Library PAUP. http://paup.csit.fsu.edu/.

[66] Library phangorn. http://cran.r-project.org/web/packages/phangorn/index.html.

[67] Library Phybase. http://cars.desu.edu/faculty/lliu/research/phybase.html.

[68] Library PHYLIP. http://evolution.gs.washington.edu/phylip.html.

[69] Library PHYLIP usage. http://evolution.genetics.washington.edu/phylip/software.serv.html#PHYLIP-servers.

[70] Library PhyloNet. http://bioinfo.cs.rice.edu/phylonet/index.html.

[71] Library PhyNav. http://www.cibiv.at/software/phynav/.

[72] Library Quartet Suite. http://genome.cs.iastate.edu/CBL/download/.

[73] Library RAxML. http://wwwkramer.in.tum.de/exelixis/software.html.

[74] Library Robinson and Foulds topological distance. http://www.bio.umontreal.ca/Casgrain/en/labo/robinson_foulds.html.

[75] Library SeqAn. http://www.seqan.de/dddoc/html/index.html.

[76] Library SeqAn article. http://www.biomedcentral.com/1471-2105/9/11.

[77] Library Supertree. http://darwin.zoology.gla.ac.uk/%7Erpage/supertree/.

[78] Library TopD/fMts. http://genomes.urv.es/topd/.

[79] Library with Jonker and Volgenant algorithm. http://tolweb.org/tree/learn/concepts/whatisphylogeny.html.

[80] MinGW webpage. http://www.mingw.org/.

[81] Neighbor joining. http://en.wikipedia.org/wiki/Neighbour_joining.

[82] Newick definition. http://en.wikipedia.org/wiki/Newick_format.

[83] Newick Phylip site. http://evolution.genetics.washington.edu/phylip/newicktree.html.

[84] Nitish Korula. Lecture materials for the Combinatorial Optimization lecture, University of Illinois http://www.cs.illinois.edu/class/sp10/cs598csc/Lectures/Lecture10.pdf, 2010.

[85] Norm definition. http://en.wikipedia.org/wiki/Norm_(mathematics)#p-norm.

[86] Phylogeny programs listing. http://evolution.genetics.washington.edu/phylip/software.html.

[87] Physics Department at the University of Illinois course materials. http://guava.physics.uiuc.edu/~nigel/courses/598BIO/498BIOonline-essays/hw3/files/hw3_li.pdf.

[88] Ryszard Riedel. Dżem. *Naiwne pytania*. http://www.dzem.com.pl/.

[89] Talko.origins Usenet newsgroup. http://www.talkorigins.org/faqs/comdesc/phylo.html#trees.

[90] tar command manual. http://man.cx/tar(1)/.

[91] Tree of Life Web Project. http://tolweb.org/tree/learn/concepts/whatisphylogeny.html.

[92] Valgrind webpage. http://valgrind.org/.

[93] A. M. Vandamme, E. J. Bernard, Y. Azad, M. Weait, and A. M. Geretti. *The use of phylogenetic analysis as evidence in criminal investigation of HIV transmission*. 2007. The report of National AIDS Trust http://www.aidsactioneurope.org/index.php?id=196&tx_windpublications_pi1[download]=128&tx_windpublications_pi1[datafield]=0.

[94] Image from. http://en.wikipedia.org/wiki/Charles_Darwin.

[95] Image from. http://en.wikipedia.org/wiki/Ernst_Haeckel.

[96] Image from. http://www.talkorigins.org/faqs/comdesc/phylo.html.

[97] Image from. http://en.wikipedia.org/wiki/Multiple_sequence_alignment.

[98] Zdzislaw Beksinski. Oil paint on a fibre-board. http://beksinski.dmochowskigallery.net/, 1975.

[99] Zdzislaw Beksinski. Oil paint on a fibre-board. http://beksinski.dmochowskigallery.net/, 1976.

[100] Zdzislaw Beksinski. Oil paint on a fibre-board. http://beksinski.dmochowskigallery.net/, 1986.

[101] Zdzislaw Beksinski. Oil paint on a fibre-board. http://beksinski.dmochowskigallery.net/, 1988.

[102] Zdzislaw Beksinski. Oil paint on a fibre-board. http://beksinski.dmochowskigallery.net/, 1988.