Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Совместимость филогенетических деревьев

Курсовая работа студента 445 группы

Катышева Алексея Александровича

Научный руководитель ……. К. В. Вяткина

к.ф.-м.н., доцент

Санкт-Петербург

2010

Оглавление

Введение.

1. Основные понятия.

Филогенетическая систематика (филогенетика) – это область более общей биологической систематики, которая занимается идентификацией и прояснением эволюционных взаимоотношений среди разных видов жизни на Земле, как современных, так и вымерших. Основателем систематики, области науки, которая занимается классификацией живых организмов и взаимоотношениями между компонентами живого, считается Карл Линней.

Одним из основных объектов филогенетики является филогенетическое дерево (эволюционное дерево, дерево жизни). Это такое дерево, которое отражает эволюционные взаимосвязи между различными видами живых организмов или другими сущностями (напр. доменами белка), имеющими общего предка. Вершины таких деревьев бывают трех типов: листья, узлы и (максимум один) корень. Листья – это терминальные вершины, каждый лист обычно отображает некоторый объект, подверженный эволюции (например, вид живых организмов). Узел – это некоторое эволюционное событие (разделение вида-предка на два или более, иногда просто промежуточное состояние). Корень (в том случае, когда он выделен) представляет собой общего предка всех рассматриваемых объектов.

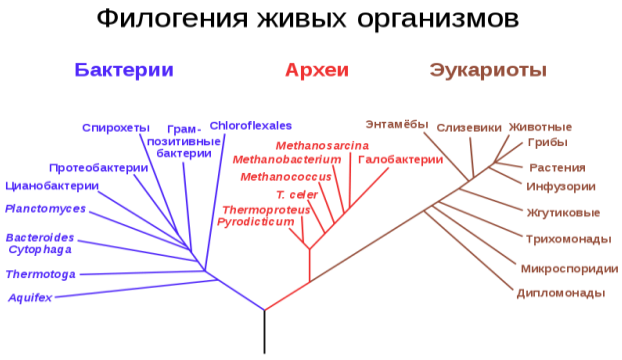

Рисунок 1

Как уже было сказано, филогенетическое дерево может быть *укоренённым* или *неукоренённым*. Рисунок 1 представляет теоретически укоренённое филогенетическое дерево, раскрашенное в соответствии с трёхдоменной системой живых организмов. Неукоренённое дерево не содержит корня и отражает взаимосвязь между листьями без предполагаемого положения общего предка. Необходимость в рассмотрении неукоренённых деревьев возникает из-за того, что часто связи между узлами восстановить легче, чем направление

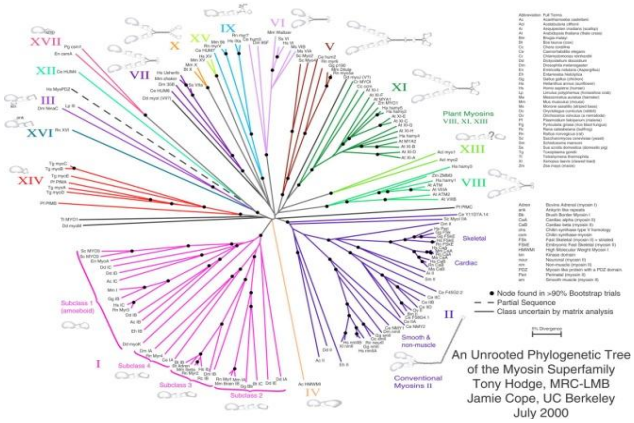

Рисунок 2

эволюции. Рисунок 2 представляет собой неукоренённое филогенетическое дерево (дерево белков-миозинов). Данная работа в основном посвящена укоренённым деревьям.

Также филогенетическое дерево может быть *бифуркационным* и *помеченным*. В бифуркационном дереве к каждому узлу подходят ровно 3 ветви (в случае укоренённого дерева – одна входящая ветвь и две исходящие). Помеченное дерево содержит названия листьев, тогда как немаркированное просто отражает топологию. В нашей работе будут рассматриваться только небифуркационные и помеченные деревья – то есть к каждому узлу могут подходить две (в случае если этот узел – промежуточный), три и более ветви, в листьях записаны названия видов, белков и т. п.

*Для того чтобы получить больше информации о филогенетических деревьях – см. Wikipedia [3]

2. Введение в проблему.

В настоящий момент существует немало различных способов построения филогенетических деревьев. Как правило, эти методы основываются на изучении последовательностей белков или нуклеиновых кислот (ДНК или РНК). Для построения филогенетического дерева бывает достаточно одного гена. Однако филогенетические деревья практически никогда не дают полного и абсолютно правильного описания эволюционных процессов. Данные, используемые тем или иным алгоритмом, содержат в себе разного рода шум. Поэтому два дерева, построенные для одного и того же набора видов по двум независимым наборам данных или при помощи двух различных способов (алгоритмов), с большой вероятностью будут в той или иной степени отличаться друг от друга. Возникает несколько логичных вопросов: насколько различающиеся деревья получены из разных исходных данных (или разными методами), и можно ли каким либо образом объединить все полученные деревья в одно.

Нас же будет интересовать следующая связанная с вышесказанным проблема теории филогенетических деревьев – так называемая *проблема совместимости деревьев* (Tree Compatibility Problem).

Изложим более формально суть проблемы. Дано некоторое множество филогенетических деревьев для одного и того же набора видов (сущностей). Требуется узнать, является ли этот набор деревьев совместимым в следующем смысле: существует ли такое *общее* дерево, которое объединяет в себе информацию все исходные деревья в той или иной степени. Часто это просто дерево, из которого можно получить любое из исходных с помощью некоторого набора операций над вершинами дерева.

К примеру, в классическом варианте проблемы совместимости деревьев допускаются операции над вершинами дерева лишь одного вида – позволяется стягивать любые два соседних узла (самая очевидная и при этом самая разумная операция над вершинами дерева). Очень долгое время лучшим полиномиальным алгоритмом решения этой проблемы был алгоритм, предложенный Гасфилдом (Gusfield) (см. [1]). Сложность этого алгоритма – $O(n^2k)$, где n – количество видов, а k – количество деревьев. Однако в 1992 году Tandy J. Warnow был предложен линейный алгоритм (т. е. O(nk)) решения этой проблемы. (Более подробно о классическом варианте Tree Compatibility Problem - в статье [2]) . На данный момент проблема в своем классическом варианте полностью исчерпана.

Единственным минусом классического подхода является то, что даже для двух довольно похожих друг на друга филогенетических деревьев общего дерева может не существовать.

В нашей работе мы рассмотрим Tree Compatibility Problem в несколько расширенном по сравнению с классическим варианте виде. Для этого мы увеличим набор возможных операций над вершинами дерева. К очевидной и вполне разумной операции стягивания двух соседних узлов мы добавим еще одну операцию, чуть менее очевидную, но при этом достаточно разумную – стягивание двух узлов, расстояние между которыми равно двум (иначе – имеющих общего соседа). Обоснуем разумность такой операции:

- Во-первых, такое стягивание не нарушает структуру дерева, то есть после выполнения такой операции дерево остается деревом (читатель может сам убедиться в справедливости этого утверждения);
- Во-вторых, эта операция разумна с точки зрения филогенетики. Например, если мы стягиваем двух сыновей одного узла (в укорененном случае), то это может означать, что в этом узле не происходило эволюционного разделения на два различных промежуточных вида. Ну а если стягивать «деда» с «сыном», то это можно объяснить *обратной мутацией*.

Также отметим, что используемые нами операции можно обобщить одной операцией - стягиванием двух узлов, расстояние между которыми не превышает двух. Также очевидно, что любая такая операция уменьшает количество узлов в дереве на 1, поэтому можно легко и корректно определить термин *минимальное общее дерево* – это общее дерево с наименьшим количеством вершин (соответственно минимальное суммарное количество операций для получения всех деревьев исходного набора)

Сформулируем цели нашей работы.

3. Цели работы

1. Теоретическая часть. Узнать, для любого ли исходного набора деревьев существует общее (в смысле введенных нами операций) дерево. В случае положительного ответа для укоренённых деревьев мы сразу же получим положительный ответ для неукоренённых деревьев, так как в любом неукоренённом дереве можно произвольным образом выделить корень (например, выбрать в качестве корня соответствующие друг другу листья). В случае отрицательного ответа было бы неплохо сформулировать критерии, при выполнении которых общее дерево существует.
2. Теоретико-практическая (алгоритмическая) часть. Выработать приемлемый по объему требуемой памяти и времени работы алгоритм для нахождения минимального (возможно, близкого к минимальному) общего дерева. В случае отрицательного ответа на вопрос первого пункта этот алгоритм будет также алгоритмом для автоматической проверки совместимости. При этом наряду со временем работы очень важным критерием будет являться объем общего дерева. Хорошим результатом будет квадратичный по количеству листьев и линейный по количеству деревьев алгоритм для нахождения минимального общего дерева. Идеальным вариантом будет линейное время работы алгоритма.
3. Практическая часть. Реализовать полученные в пункте 2 алгоритмы с приемлемым временем работы, провести тесты на реальных примерах.
4. Заключение. Оценить результаты, сформулировать открытые проблемы и/или гипотезы для дальнейших исследований.

1. Теоретические основы

Проблема совместимости деревьев

Ранее было отмечено, что одной лишь операции стягивания соседних вершин недостаточно для того, чтобы любой набор деревьев были совместимым. С добавлением еще одной операции множество совместимых наборов очевидно расширится. Вопрос лишь в том, насколько шире оно станет. Ответ на этот вопрос изложен ниже.

В ходе исследований было установлено, что вопрос совместимости филогенетических деревьев имеет положительный ответ для любого набора исходных деревьев (очевидно, что мы рассматриваем исключительно *однотипные* деревья - такие деревья, листья которых помечены с использование одного и того же набора меток). Тем самым множество совместимых наборов максимально расширилось и теперь совпадает с множеством допустимых наборов. Сформулируем и докажем этот факт.

Теорема: для любого набора однотипных укоренённых деревьев T_1, T_2, … T_k существует такое укоренённое дерево. T, что любое из деревьев исходного набора можно получить из T за некоторое количество шагов при помощи операции стягивания двух вершин, находящихся на расстоянии не более 2.

Доказательство будет состоять из двух частей: в первой части будет явно построено некоторое дерево, а во второй части будет доказано, что это дерево является искомым.

Построение: пусть r_1, r_2, …, r_k – корни соответствующих деревьев, а v_1, v_2, …, v_n – набор листьев каждого дерева (каждая v_i обозначает сразу k листьев, по одному в каждом из деревьев, а также лист в строящемся дереве). Обозначим для любого i от 1 до n за $d_i = \max(dist(r_j, v_i))$ по всем j от 1 до k, то есть максимальная по всем деревьям глубина листа v_i (глубина = количество ребер на пути от корня до листа). Теперь возьмем будущий корень r и подвесим каждый лист v_i к корню на цепочку вершин длины d_i (для каждого листа от корня идет отдельная цепочка), то есть так, чтобы глубина листа v_i была равна d_i и пути от корня к разным листьям имели лишь одну общую точку – корень r. Получили дерево T.

Достаточность: рассмотрим T_1, опишем процедуру сведения дерева. T к T_1, остальные деревья из исходного набора - аналогично. Дерево в промежуточном состоянии обозначим за T’. Сначала в дереве T сократим длину цепочки, идущей к каждому листу, до значения глубины этого листа в дереве T_1. Далее рассмотрим первый уровень дерева T_1 (то есть всех сыновей корня). Каждому из узлов этого уровня сопоставим множество листьев, которые лежат в поддереве этого узла. Для каждого такого множества листьев L сделаем так, чтобы вершины первого уровня дерева T’, лежащие на пути от корня к

листьям из L, объединились в одну. Это легко сделать, так как расстояние между любыми двумя из этих вершин ровно 2 (т. к. есть общий сосед – корень дерева). Таким образом мы привели в порядок первый уровень дерева T'. Далее проделаем то же самое с уровнем 2, 3, и так далее. На каждом уровне расстояние между объединяемыми вершинами будет ровно 2, так как у них будет общий сосед с предыдущего уровня.

■

Следствие: для любого набора однотипных деревьев T_1, T_2, … T_k существует такое дерево T, что любое из деревьев исходного набора можно получить из T за некоторое количество шагов при помощи операции стягивания двух вершин, находящихся на расстоянии не более 2.

Доказательство: очевидно, искусственно укореним все исходные деревья.

Замечание: общее дерево T, полученное в Теореме, не обязательно является минимальным общим деревом. В тех случаях, когда оно является минимальным, можно смело утверждать, что исходный набор деревьев несовместим.

2. Алгоритмы

В Теореме из предыдущего раздела обосновано существование общего дерева. Замечание говорит нам о том, что построенное в данной Теореме дерево не несет в себе никакого практического смысла Отсюда очевиден факт существования минимального общего дерева. Но теорема ничего не говорит нам о том, как можно найти минимальное общее дерево.

Для начала приведем несколько очень важных наблюдений:

1. Длина пути от корня до каждого листа не может быть меньше чем в дереве T. Это очевидно, так как наши операции могут лишь сократить это расстояние. Таким образом, уменьшить общее дерево можно лишь за счет объединения вершин из разных цепочек.
2. Очевидно, что если во всех исходных деревьях цепочки, спускающиеся к листьям v и w, совпадают по некоторому количеству вершин и расходятся только ближе к листьям, то и в дереве T цепочки, идущие к этим листьям, можно сразу объединить по нескольким звеньям. Вопрос лишь в том, сколько звеньев этих цепочек можно объединить.

Далее введем несколько понятий.

Определение: дерево назовем *приемлемым*, если в нем максимально объединены звенья цепочек, ведущих к листьям. Это означает, что при объединении любых двух звеньев (имеются ввиду две вершины с одинаковым родителем, расстояние между которыми ровно 2) дерево перестает быть общим.

В ходе исследований удалось разработать алгоритм построения приемлемого дерева. Для описания алгоритма нам понадобится ввести еще одно понятие.

Определение: матрица целых чисел размера n×n называется *дескриптором* дерева T, если:

- Каждому листу соответствует число от 1 до n, а также строка с этим номером и столбец с этим номером;
- В каждой диагональной клетке записана глубина соответствующего листа;
- В ячейке с координатами (i, j) записана длина общего пути от корня к листьям, соответствующим i и j (глубина последнего от корня общего узла цепочек к этим листьям)

Очевидно, что каждое дерево однозначно задает дескриптор. Также очевидно, что по дескриптору однозначно строится дерево (однако не любая матрица является дескриптором).

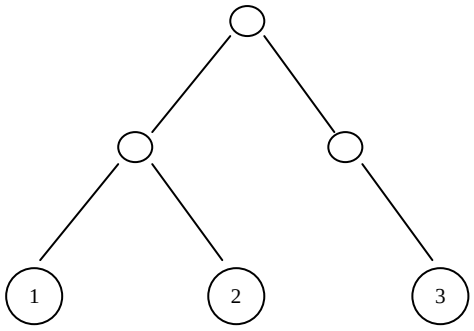

Рисунок 3

2	1	0
1	2	0
0	0	2

Таблица 1

Матрица из Таблицы 1 представляет собой пример дескриптора, соответствующего дереву, изображенному на рисунке 3.

2	2	0
2	1	0
0	0	1

2	1	1
1	3	0
1	1	2

2	1	1
1	2	0
1	0	2

Таблица 2

В Таблице 2 приведены примеры матриц, которые по различным причинам не являются дескрипторами. В первом случае диагональный элемент меньше некоторого элемента из той же строки. Во втором случае матрица несимметрична относительно главной диагонали. В третьем случае не соблюдается условие транзитивности (это условие выглядит так – $M_{ij} >= \min(M_{ik}, M_{kj})$ для любых i, j и k).

Итак, схема алгоритма построения приемлемого дерева состоит из 3 этапов:

1. Для каждого исходного дерева составляется дескриптор
2. По полученным дескрипторам строится еще один – дескриптор приемлемого дерева
3. Полученный дескриптор преобразуется в приемлемое дерево

2.1. Построение дескриптора

В данном разделе мы рассмотрим первый этап алгоритма, который предусматривает построение дескриптора данного филогенетического дерева.

Входные данные: филогенетическое дерево с известным набором пометок на листьях.

Выходные данные: матрица n×n, которая является дескриптором исходного дерева.

Алгоритм по шагам:

1. Создаем пустую таблицу

2. Запускаем рекурсивный метод заполнения таблицы с дополнительным параметром depth (корню передаем depth = 0)
3. Если текущая вершина - лист, то записываем в соответствующую данному листу диагональную ячейку значение depth
4. Если же текущая вершина - узел, то для каждой пары листьев, пути к которым идут в разные поддеревья, в соответствующие этой паре ячейки таблицы записываем значение depth. Запускаем рекурсивно метод для каждого сына, при этом в качестве дополнительного параметра передаем значение depth + 1.

Очевидно, что после полного прохода алгоритма в каждую ячейку будет записано некоторое значение ровно 1 раз, тем самым время работы алгоритма – $O(n^2)$. Но это при условии, что множество листьев в поддереве получаем за O(1). Этого легко добиться за счет дополнительной памяти, надо всего лишь в каждом узле дополнительно хранить список листьев в поддереве данного узла (потребуется порядка Trace(descriptor) дополнительной памяти, так как каждый лист надо хранить столько раз, какова глубина этого листа) . Также очевидно, что для работы алгоритма требуется память порядка $O(n^2)$.

Замечание: можно сэкономить на памяти, если не хранить все те нули, которые будут записаны в дескрипторе. Этого легко добиться, используя не массивы, а так называемые хэш-карты, при этом нули просто не нужно заносить в хэш-карты. Именно так эта часть алгоритма и будет реализована.

2.2. Восстановление дерева по дескриптору

В данном пункте рассмотрен третий этап алгоритма.

Входные данные: матрица n×n, которая является дескриптором некоторого филогенетического дерева.

Выходные данные: филогенетическое дерево, соответствующее данному дескриптору.

Алгоритм:

1. По значениям, записанным в диагональных ячейках, создается дерево, описанное в Теореме
2. Для каждой недиагональной ячейки соответствующие ей цепочки к листьям объединяются на такое число звеньев, какое записано в данной ячейке.

Очевидно, что время работы есть O(Trace(descriptor)). Диагональных элементов n, каждый из них не превосходит n + c (в случае если в исходных деревьях было слишком

много промежуточных узлов, перед началом работы алгоритма их можно было предварительно удалить). В общем случае время работы будет O(n· maxLeafDepth).

Также для работы алгоритма требуется память порядка O(n· maxLeafDepth + n^2)

2.3. Построение общего дескриптора

В данном пункте мы приведем алгоритм построения общего дескриптора для 2х дескрипторов (второй этап вышеописанного алгоритма). Далее мы покажем, что алгоритм легко расширяется до любого количества входных дескрипторов.

Входные данные: дескрипторы филогенетических деревьев (далее D и E).

Выходные данные: дескриптор общего дерева (далее F)

Введем дополнительные обозначения:

- $D_i = D(i, i)$, $E_i = E(i, i)$, $F_i = F(i, i)$,
- $C_{kn} = \max(D_k – D(k, n), E_k – E(k, n))$ - это максимум из двух расстояний от «вершин расхождения» (для листьев k и n) до листа k.

Алгоритм:

1. Сначала заполняются диагональные ячейки в соответствии с Теоремой. Формально: $F_i = \max(D_i, E_i)$, $i = 1…n$;
2. Далее находятся значения недиагональных элементов. Основной принцип состоит в том, что расстояние от «вершины расхождения» двух цепочек до соответствующих листьев должно быть минимально возможным. Формально значение диагональной ячейки вычисляется так: $F(i, j) = \min(F_i - C_{ij}, F_j - C_{ji})$, $i = 1…n$, $j = 1…n$, $i \neq j$;

Замечание: легко обобщается – на месте D и E может быть любое количество дескрипторов (изменится только количество аргументов у max во всех трех случаях)

Замечание: доказано, что:

- полученный дескриптор является корректным дескриптором,
- этот дескриптор является дескриптором искомого приемлемого дерева,

но не очень сложные, громоздкие и отчасти переборные доказательства этих фактов здесь приведены не будут.

Кратко ход доказательства:

- симметричность дескриптора и доминирование диагональных элементов – очевидно, надо всего лишь проверить транзитивность $F_{ij} >= \min(F_{ik}, F_{kj})$ для любых i, j и k.
- очень просто проверяется тот факт, что нельзя более объединить никакие два звена; остается только проверить «сводимость» дерева к каждому из исходного набора.

Очевидно, что время работы алгоритма – $O(n^2 \cdot k)$. Это объясняется тем, что всего n^2 ячеек, каждая заполняется за время порядка O(k). Требования по памяти также порядка $O(n^2 \cdot k)$.

2.4. Оценка ресурсов алгоритма

В данном пункте мы оценим объем требуемой памяти и суммарное время работы приведенного выше алгоритма.

Время работы:

$$\Sigma = k \cdot O(n^2) + O(n^2 \cdot k) + O(n \cdot maxLeafDepth) = O(n \cdot (2nk + maxLeafDepth))$$

Таким образом, если в деревьях разумное (O(n)) количество промежуточных вершин, то приведенный выше алгоритм является квадратичным, то есть $O(n^2 \cdot k)$. При этом общее дерево строится всегда и имеет достаточно приемлемый размер.

Общий объем требуемой памяти:

$O(n^2 \cdot (k+1)) + O(n \cdot maxLeafDepth) = O(n \cdot (nk + n + maxLeafDepth))$.

Первое слагаемое – необходимая для хранения всех дескрипторов память. Второе слагаемое – оценка сверху на объем филогенетического дерева.

3. Реализация

Для создания библиотеки для работы с филогенетическими деревьями, включающей в себя реализацию описанного в предыдущей части алгоритма, была выбрана платформа Java SE по следующим причинам:

- Удобство создания объектно-ориентированной библиотеки
- Широкий набор уже разработанных библиотек (Java SDK), в том числе отлично реализованная библиотека коллекций
- Кроссплатформенность

3.1. Описание классов

1. BaseNode – абстрактный класс для описания вершин дерева. Содержит в себе ссылку на родителя, а также объявление методов fullDescriptor(…) и getLeafs()
2. Node – класс, описывающий узел дерева. Хранит в себе список сыновей и набор листьев в поддереве данного узла, а также набор методов для работы с узлами. Реализован рекурсивный метод fullDescriptor(…) в соответствии с описанным выше алгоритмом:

```
public void fullDescriptor(Map<String, Map<String, Integer>> des, int depth)
{
	for (BaseNode child: myChildren) {
		child.fullDescriptor(des, depth+1);
	}
	if (depth == 0) return;
	for (BaseNode child1: myChildren) {
		for (BaseNode child2: myChildren) {
			if (child1 != child2) {
				for (Leaf leaf1: child1.getLeafs()) {
					for (Leaf leaf2: child2.getLeaves()) {
						String m1 = leaf1.getMark();
						String m2 = leaf2.getMark();
						des.get(m1).put(m2, depth);
						des.get(m2).put(m1, depth);
					}
				}
			}
		}
	}
}
```

3. Leaf – класс, описывающий лист дерева. Хранит в себе пометку этого листа. Реализованы методы getLeafs() и fullDescriptor(…) очевидным образом и в соответствии с приведенным выше алгоритмом:

```
public void fullDescriptor(Map<String, Map<String, Integer>> des, int depth){
	des.get(myMark).put(myMark, depth);
}
```

4. PTree – класс, непосредственно представляющий филогенетическое дерево. Хранит в себе корень и отображение меток в листья дерева. Реализованы методы getDescriptor(), а также методы объединения дескрипторов и построения дерева по дескриптору. Все реализовано в соответствии с приведенным выше алгоритмом. Приведем реализацию метода getDescriptor():

```
private Map<String, Map<String, Integer>> getDescriptor() {
      Map<String, Map<…>> des = new HashMap<String, Map<…>>();
      Set<String> marks = myLeafsByMarks.keySet();
      for (String m1: marks) {
            Map<String, Integer> line = new HashMap<String, Integer>();
            des.put(m1, line);
      }
      myRoot.fullDescriptor(des, 0);
      return Collections.unmodifiableMap(des);
}
```

3.2. Тестирование

Алгоритм был протестирован как на искусственных, так и на реальных примерах различного размера. Полученные результаты при k = 2 приведены в таблице:

Количество листьев	Количество симуляций	Время работы, сек
10	10 000	2
44	1 000	3
90	1 000	12
512	10	9
1024	10	41

Таблица 3

Как видно из Таблицы 3, асимптотика $O(n^2)$ сохраняется (при увеличении объема входных данных в 2 раза время работы алгоритма увеличивается примерно в 4 раза).

Заключение

В данной работы была рассмотрена проблема совместимости филогенетических деревьев в неклассическом варианте. Расширив набор операций над вершинами дерева, удалось добиться того, что любой набор деревьев является совместимым и для него можно найти общее дерево. Также был разработан и реализован алгоритм построения некоторого приемлемого дерева для случая укоренённых деревьев. Конечно, в некоторых случаях это дерево будет довольно велико, но этот факт будет сигнализировать о том, что исходный набор деревьев слабо совместим.

Направления дальнейших исследований:

1. Проверить на истинность гипотезу о том, что приемлемое дерево является минимальным. В данной работе нам не удалось ни доказать, ни опровергнуть этот факт.
2. Рассмотреть случай неукоренённых деревьев, проверить полиномиальность задачи о нахождении минимального приемлемого дерева в этом случае.
3. Возможно ли нахождение приемлемого дерева за линейное время?

Список литературы

1. D. Gusfield, “ Efficient algorithms for inferring evolutionary trees” // Networks, vol. 21, pp. 19-28 (1991)
2. Tandy J. Warnow, “Tree Compatibility and Inferring Evolutionary History” // Journal of Algorythms, vol. 16, pp. 388-407 (1994)
3. Wikipedia, the free encyclopedia, “Phylogenetic Tree” (http://en.wikipedia.org/wiki/Phylogenetic_tree)