一个基于树编辑距离及其相关问题的调查

作者：Philip Bille （这是一个基于欧洲联合 IST-2001-35443 IST 项目支持的 DSSCV 部分工程的文献）

摘要：

我们通过删除，插入，和重新标记节点这些简单的本地操作来调查比较标签树的问题。通过这些操作，会引发树编辑距离,调整距离,和其包容的问题。对于这些问题我们审查了已存可用的和现存出现的结果，详细地，用一个或多种核心算法去解决这个问题。

关键字：树匹配，编辑距离

引子介绍：

树状结构是在电脑科学里是最常见并且被充分研究了的组合结构。特别是，比较树状结构的问题发生在好几个不同的区域里，比如计算机生物学，结构化文本数据库，图像分析，自动定理证明，以及编译程序优化。例如，在计算机生物学中，在多种距离措施下比较树状结构与树状结构之间的相似性被用在 RNA 的第二等结构比较中。

让 T 作为一个根树。如果每个节点都是从一个确定的有穷字母 Σ 赋值而来的符号，我们将 T 称之为一个有标号树。如果在这个 T 的同级兄弟树中，存在从左到右的顺序，我们将这个 T 称之为一个有序树。在本文中我们考虑了通过在标记树上做出一些简单原始的的操作去匹配问题。如果 T 是一个有序树，这些操作就可以被如下定义：

重新标记 改变在 T 里一个节点 v 的标签

删除 删除一个 T 中父节点是 v'的非根节点 v，将 v 的子节点变成 v'的子节点。这个子节点将替代 v，保证原有 v'孩子们的左右排序。（p1 完）

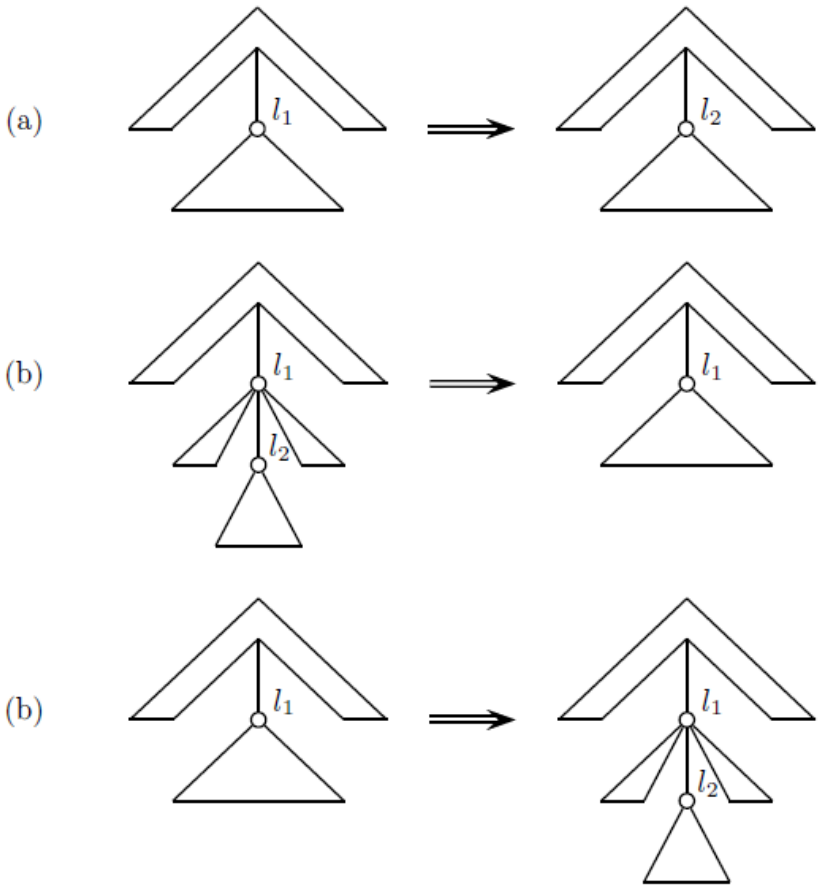

图 1：a) 将节点标记 l1 重新标记为 l2。 b) 删除标记过的 l2 节点。c) 将标记为 l2 的节点插入，使其作为 l1 的子节点。

插入 删除的补充。插入一个节点 v 作为 v'在 T 中的一个子节点，使 v 成为 v'子节点的同时保证其孩子节点的排序。
图 1 解释阐明了它的操作。对于无序树来说，操作也是相似的。在这种情况下，插入和删除的操作是在子集上而不是子序列上。我们在编辑操作上定义了三个问题。将 T1 和 T2 成为标记树（有序或无序）。

树编辑距离：假设我们现有一个定义在每个编辑操作上的花销函数。T1 与 T2 间的编辑脚本 S 为将 T1 转化为 T2 编辑序列操作。S 的花销是在 S 操作中的花销总数。T1 与 T2 之间最好的编辑脚本，是 T1 与 T2 间最小花销的编辑脚本，同时这个花销也是这个树编辑距离。这个树编辑距离的问题所在是计算编辑距离和其相应的编辑脚本。（p2 完）

树对齐距离：假设我们现有一个定义在一对标签上的花销函数。从 T1 和 T2 获得的对齐操作 A 如下。首先，我们把有空格标记的节点插入 T1 与 T2，使得他们在忽略本身标签时可以成为同构。接着把所得到的树按顶点对齐的方式重叠，最后得到的这个对齐 A 就是每个节点由一对标签标记的树。A 的花费是 A 中每对相对标签花费的总数。T1 和 T2 的最好的

对齐就是最小花费的对齐，这个花费被称为 T1 和 T2 的对齐距离。对齐距离的问题是去计算对齐距离和一个相应的对齐操作。

树包含：T2 包含 T1，并且只有在 T2 中删除节点才能获得 T1。树包含问题是定义 T1 是否包含在 T2 内。

在这篇文章里，我们会调查这些问题中的每一个，并讨论从它们中得到的结果。仅供参考，在第 27 页的表格 1 总结了大多数的可用结果。本文包含了所有这些问题，并附带讨论一些其它的。树编辑距离问题是最常见的问题。对齐距离对应于一种被限制了的编辑距离，而树包含是一种包括了编辑和对齐距离问题的特殊情况。除了这些简单的关系，我们也学习研究了一些在编辑距离上有趣的变化，致使它现在成为了一个更复杂的画面。

我们审查了有序和无序版本的问题。对于无序的情况，我们发现最终导致所有的问题一般是 NP 难题。甚至于，树编辑距离和对齐距离问题都是 MAX SNP 难题[4]。然而，多项式时间算法却可用于各种有趣的限制和一些特殊的情况。比如说，多项式时间算法可以用在当我们对无序树编辑距离问题施加结构保留限制，使得不相交子树被映射到不相交子树时。并且，用这样的方法还可以有效解决恒定度树的无序对齐问题。

有序版本的问题多项式时间算法已经存在。这些都是基于动态规划的基础技术（见，比如[9, 章节 15]），其中大部分都是简单的组合算法。然而就在最近，快速矩阵乘法，一种更高级的技术已经能成功使用在树编辑距离问题上了[8]。

这个调查用了以下方式来处理这些问题。我们对于每个问题及其变体，都检查了有序和无序版本的结果。在大多数的情况下会包括一个对问题的正式定义，一个可用结果的比较和一个用于取得结果的技术的描述。更重要的是，我们还会针对每个问题选择一个或多个核心算法并几乎完全地展现细节。具体来说，我们将描述算法，证明它是正确的，并分析其时间复杂度。为了简洁，我们将省略几个引理的证明并跳过一些不太重要的细节。在大多数情况下，详细给出的算法是更高级算法的基础。通常大多数的算法，用于上述问题之一的是相同动态程序的改进。（p3 完）

本次调查的主要技术贡献是将问题和算法展现在一个常规框架里。我们希望，这能够使得读者能够对于这些问题以及它们之间的关联性得到一个更好的概念以及更深层次的理解。在某些文献中，对于展现问题存在着一些差异。比如，Klein [25] 用一种对边运算操作的方式使用了编辑操作考虑了有序编辑距离的问题。他提出一个使用减法的算法，用来解决括号对字符串问题。相比之下，Zhang 和 Shasha [55] 给出了基于树的后序遍历的算法。事实上，如果考虑正确的设置，这些算法有着很多共同的特征。在本文中，我们将在这两个方式之间建立一个新的框架来桥接展现这些算法。

文献中出现的另一个问题是关于编辑距离问题的定义缺乏一致性。而这里给出的定义是我们觉得最自然也是最精通的。然而，我们已经考虑了用大相径庭的距离测量方式得到好几个可选的结局[30, 45, 38, 31]。我们审查了一些变体并将之与我们的定义比较。应该要知道的是，这里定义的编辑距离问题有时被称为树到树校正问题。

这个调查采用了一个理论上的观点。然而，上述问题不仅是有趣的数学问题，而且它们也出现在许多实际情况中，并且我们认为它对于能在现实生活问题中表现良好的算法也起到了重要的作用。有关于实际问题，请参考，比如，[49, 46, 40]。

我们把注意力限制在顺序算法上。然而，对于一些关于编辑距离问题的并行算法已经存在了一些研究，比如，[55, 53, 41]。

这总结了本文的内容。由于比较树的基本性质及其许多的应用，我们已经设计了一些其他用来比较树与树的方法。在本文中，我们选择了局限出一小部分我们能够详细说明描述的问题。其他问题包括树模式匹配 [27, 10],和 [16, 35, 56],最大协议子树[19, 11]，最大公共子树[2, 20]和最小公共子树[34, 13]。

1.1 大纲

在第二部分我们给出了一些初步性。在第 3，第 4 和第 5 章节，我们分别调查研究了树编辑距离，对齐距离和包含问题。在第 6 章节中，我们总结了一些开放性问题。

2. 正文前的书页和符号

在这个章节中，我们把在整篇文章中将会使用的符号和定义做了定义。对于图 G，我们分别用 V(G)和 E(G)表示节点和边的集合。（p4 完）将 T 作为一个根树。T 的根是由 root(T)来表示的。T 的尺寸，由|T|来表示：即|V (T)|。节点 v 的深度 v ∈ V(T), 即 depth(v)，是从 v 到根(T)路径上边的数量。节点 v 的度数，deg(v)是 v 的子节点数。我们将这些定义进行了扩展，使得 depth(T)和 deg(T)分别表示 T 中任何节点的最大深度和度。没有子节点的节点是一个叶子或是一个内部的节点。叶子的叶数由 leaves(T)表示。我们用 parent（v）表示节点 v 的父节点。如果两个节点具有相同的父节点，则它们是兄弟节点。对于 T1 和 T2 这两个树，我们将通过 Li，Di 和 Ii 来表示 leaves(Ti)，depth(Ti)和 deg(Ti)，其中 i=1,2。

令 θ 表示空树，并且使 T（v）表示在节点 v∈V（T）处作为根的 T 的子树。如果 w∈V(T(v))则 v 是 w 的祖先，如果 w∈V(T(v))\{v}，则 v 是 w 的独有祖先（即 w 属于 T(v)下某一节点但不是 v）。如果 v 是 w 的一个（独有）祖先，那么 w 是 v 的一个（独有）后代。如果在给出的兄弟节点中是从左到右的，那么这个树 T 是有序的。对于具有根 v 和子节点 v1, . . . , vi 的有序树 T，为了获得 T(v)的前序遍历，通过访问 v 然后递归地访问 T(vk)，1≤k≤i。类似地，后序遍历首先是通过访问 T(vk)，1≤k≤i，然后再访问 v。节点 w∈T(v)的前序号和后序号，由 pre(w)和 post(w)来表示，分别是在 T 前序遍历中 w 之前的节点数和后序遍历中 w 之后的节点数，表示的是分别在 T 的前序和后序遍历中在 w 之前的节点的数量。T 中的 w 左边的节点是节点 u∈V(T)的集合，使得 pre (u)< pre (w)以及 post (u)< post (w)。如果 u 在 w 的左边，那么 w 即在 u 的右边。

一片森林是一组树。如果给定在树中从左到右的顺序并且每个树是有序的，则森林 F 被排序。令 T 是有序树并且使 v∈V(T)。如果 v 有子节点 v1, . . . , vi，定义 F(vs, vt), 其中 1≤s≤t≤i，作为森林 T(vs), . . . , T (vr)。为了方便，我们设定 F(v) = F(v1, vi)。

在整篇论文中，我们假定分配给节点的标签从有限字母表 Σ 中选择。令 $\lambda \notin \Sigma$ 表示一个特殊的空白符号并定义$\Sigma_\lambda = \Sigma \cup \lambda$。我们经常在成对的标签上定义一个花销函数，$\gamma : (\Sigma_\lambda \times \Sigma_\lambda) \backslash (\lambda, \lambda) \to \mathbb{R}$。我们会经常在文中假设 γ 是一个距离度量。也就是说，对于任何 $l_1, l_2, l_3 \in \Sigma_\lambda$，满足以下条件：

1. $\gamma(l_1, l_2) \geq 0$, $\gamma(l_1, l_1) = 0$.
2. $\gamma(l_1, l_2) = \gamma(l_2, l_1)$.
3. $\gamma(l_1, l_3) \leq \gamma(l_1, l_2) + \gamma(l_2, l_3)$.

3. 树编辑距离

在本章节中，我们调查了树编辑距离问题。假设我们给出了在每个编辑操作上定义的成本函数。两个树 T1 和 T2 之间的编辑脚本 S 是将 T1 转变为 T2 的编辑操作的序列。S 的花销是在 S 操作中的花销总和。T1 和 T2 之间最佳的编辑脚本是 T1 和 T2 之间的最小成本的编辑脚本。（P5 完）这个花销被称为树编辑距离，由 δ(T1, T2)表示。编辑脚本的示例如图 2 所示。

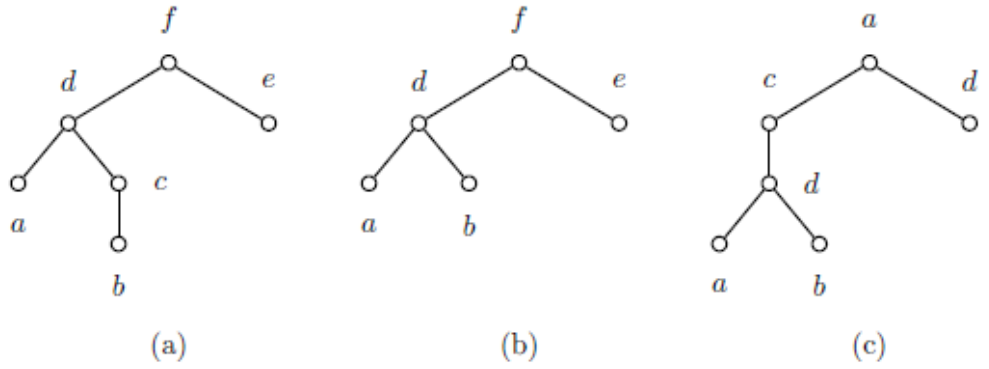

图 2：通过编辑操作将(a)转换为(c)。(a) 初始树。(b) 删除标记为 c 的节点后的树。(c) 插入节点标记为 c 和重新标记 f 到 a 和 e 到 d 后的树。

本章节其余部分如下。首先，在第 3.1 章节，我们提出一些初步性和正式定义的问题。在第 3.2 章节中，我们调查了为有序编辑距离问题获得的结果，并提出了两个当前最好的算法。该问题的无序版本在 3.3 章节中进行了讨论。在第 3.4 章节中，我们回顾了施加在各种结构保留约束时编辑距离问题的解决方法。最后，在第 3.5 章节中，我们考虑了一些问题上的其他变体。

3.1 编辑操作和编辑映射

令 T1 和 T2 成为被标记树。接下来[43]我们用(l1→l2)表示每个编辑操作，其中 $(l_1, l_2) \in (\Sigma_\lambda \times \Sigma_\lambda) \backslash (\lambda, \lambda)$。如果 $l_1 \neq \lambda$ 并且 $l_2 \neq \lambda$，那么操作即为重新标记，如果 $l_2 = \lambda$，那么操作即为删除，如果 $l_1 = \lambda$，那么操作即为插入。我们扩展符号，使得对于节点 v 和 w 的(v→w)表示(label(v)→label(w))。当带着标签的时候，这里的 v 或 w 可以是 λ。我们通过在给定在成对标签上定义的度量成本函数 γ 设置 $\gamma(l_1 \to l_2) = \gamma(l_1, l_2)$ 来定义编辑操作的花销。操作序列 $S = s_1, \ldots, s_k$ 的成本由 $\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$ 给出。T1 与 T2 之间的编辑距离 $\delta(T_1, T_2)$ 被正式定义为如下：

$\delta(T_1, T_2) = \min\{\gamma(S) \mid S$ 是将 T1 变换为 T2 的操作序列$\}$。由于 γ 是距离度量，因此 δ 也变为距离度量。

T1 和 T2 之间的编辑距离映射（或仅仅是映射）表示的是编辑操作，被使用在很多树编辑距离问题的算法中。形式上，将三元组(M, T1, T2)定义为从 T1 到 T2 映射的有序编辑距离，如果 $M \subseteq V(T_1) \times V(T_2)$ 并且对于任何组 $(v_1, w_1), (v_2, w_2) \in M$ ：

1. $v_1 = v_2$ iff $w_1 = w_2$. （一对一条件）
2. v_1 是 v_2 的祖先，iff w1 是 w2 的祖先。（祖先条件）
3. v_1 在 v_2 的左边，iff w1 在 w2 的左边。（兄弟条件）

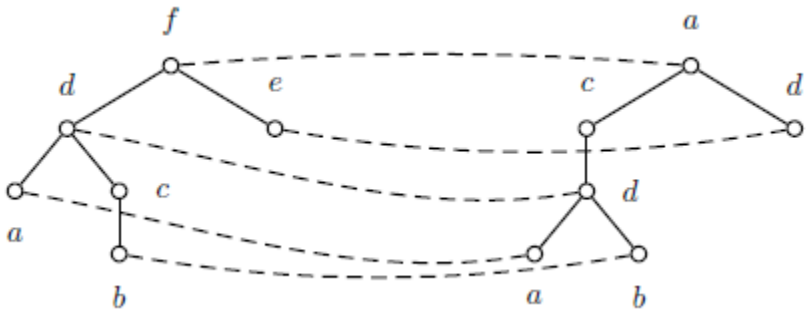

图 3：编辑脚本对应的映射如图 2 所示。

图 3 图示出了对应于图 2 中的编辑脚本的映射。我们将两个无序树之间的无序编辑距离映射的定义视为相同的，但没有兄弟条件。 当没有混乱时，我们将使用 M 代替(M, T1, T2)。令(M, T1, T2)作为映射。 当 v 出现在 M 中的一些组对里的时候，我们说在 T1 或 T2 中的节点 v 被在 M 中的一根线触及。让 N1 和 N2 分别是 T1 中的节点和 T2 中的节点，它们不被 M 中的任何线接触。M 的花销由下式给出：

$$\gamma(M) = \sum_{(v,w) \in M} \gamma(v \to w) + \sum_{v \in N_1} \gamma(v \to \lambda) + \sum_{w \in N_2} \gamma(\lambda \to w)$$

映射可以被组成。 令 T1，T2 和 T3 成为标记树。令 M1 和 M2 分别是从 T1 到 T2 和 T2 到 T3 的映射。定义$M_1 \circ M_2 = \{(v,w) \mid \exists u \in V(T_2)$，使得$(v,u) \in M_1$并且 $(u,w) \in M_2\}$。

利用这个定义，我们很容易就能得出$M_1 \circ M_2$自身变成了 T1 到 T3 的映射。由于 γ 是度量，不难看出最小成本映射等同于编辑距离：

$\delta(T_1,T_2) = \min\{\gamma(M) \mid (M,T_1,T_2)$ 是一个编辑距离映射$\}$

因此，为了计算编辑距离，我们可以计算最小成本映射。 我们将编辑距离的定义扩展到森林。也就是说，对于两个森林 F1 和 F2，δ(F1，F2)表示的是 F1 和 F2 之间的编辑距离。这个操作是被定义在树的情况下的，然而，现在可以删除森林中的树的根，并且可以通过插入新的根来合并树。 映射的定义以相同的方式扩展。

3.2 普通有序的编辑距离

有序编辑距离问题是被 Tai[43]以熟知的字符串编辑距离问题[48]泛化所引入。(P7 结束）Tai 使用$O(|T_1||T_2||L_1|^2|L_2|^2)$时间和空间来呈现有序版本的算法。后来，Zhang 和 Sasha[55]使用了一个简单的算法来改善$O(|T_1||T_2|\min(L_1,D_1)\min(L_2,D_2))$时间和$O(|T_1||T_2|)$空间的边界。Klein[25]以在相同的空间界限下得到一个比较好的最坏情况$O(|T_1|^2|T_2|\log|T_2|)^1$的时间界限为前提修改了该算法（由于编辑距离是对称的，所以实际上这个界限是$O(\min(|T_1|^2|T_2|\log|T_2|,|T_2|^2|T_1|\log|T_1|)$，为了简洁起见，我们用了缩短的形式表达）。我们将在下方详细介绍后两种算法。最近，Chen[8]提出了一种使用$O(|T_1||T_2|+L_1^2|T_2|+L_1^{2.5}L_2)$时间和$O((|T_1|+L_1^2)\min(L_2,D_2)+|T_2|)$空间的算法。对于某些种类的树，算法改进了先前的界限。 该算法使用快速矩阵乘法的结果，比上述所有算法更复杂。 需要注意的是，在上述界限中，因为距离是对称的，我们可以用 T1 交换 T2。

3.2.1 一个简单的算法

我们首先提出一个简单的递归式，它将成为我们将在下面两节中介绍的两个动态规划算法的基础。我们将只显示如何计算编辑距离。相应的编辑脚本很容易可以在相同的时间和空间边界内获得。该算法是来自 Klein[25]。然而，我们应该注意，这里给出的演示有些不同。我们认为我们的框架更简单，并能为以前的工作提供更佳的连接。

令 F 成为森林，v 是 F 中的节点。我们用 F-v 表示通过从 F 中删除 v 而获得的森林。此外，将 F-T(v)定义为通过删除 v 和 v 的所有后代而获得的森林。以下引理提供了一种为一般情况

下的森林计算编辑距离的方法。

引理 1 令 F1 和 F2 为有序森林并且 γ 是在标签上定义的度量花销函数。令 v 和 w 分别是 F1 和 F2 中最右侧的根（如果有的话）。我们得到，

$$\begin{aligned}
\delta(\theta,\theta) &= 0\\
\delta(F_1,\theta) &= \delta(F_1 - v,\theta) + \gamma(v \to \lambda)\\
\delta(\theta,F_2) &= \delta(\theta,F_2 - w) + \gamma(\lambda \to w)\\
\delta(F_1,F_2) &\models \min \begin{cases}
\delta(F_1 - v,F_2) + \gamma(v \to \lambda)\\
\delta(F_1,F_2 - w) + \gamma(\lambda \to w)\\
\delta(F_1(v),F_2(w)) + \delta(F_1 - T_1(v),F_2 - T_2(w)) + \gamma(v \to w)
\end{cases}
\end{aligned}$$

证明。前三个方程是自然而然的。为了显示最后一个方程需要考虑 F1 和 F2 之间的最小成本映射 M. v 和 w 有三种可能性：

案例 1：v 并没有被线接触。那么 $(v,\lambda) \in M$ 且最后一个方程第一种情况可适用。（P8 结束）

案例 2：w 并没有被线接触。那么 $(\lambda,w) \in M$ 且最后一个方程的第二种情况可适用。

案例 3：v 和 w 都被线触及。我们表示，这意味着(v,w) ∈M。假设(v,h)和(k,w)都在 M 里。如果 v 在 k 的右边，则 h 必须因为兄弟状态在 w 的右边。如果 v 是 k 的独有祖先，则 h 必须是祖先条件下 w 的独有祖先。这两种情况同时出现是不可能的，因为 v 和 w 是最右边的根也因此 $(v,w) \in M$。通过映射的定义，等式如下。

引理 1 提出了一个动态的程序。(F1, F2)的价值取决于较小尺寸的子问题的常数。因此，我们可以按照增加的大小顺序计算（S1，S2）的所有子问题对 S1 和 S2 来进行(F1，F2)的计算。每个新的子问题可以在恒定时间内计算。所以，时间复杂度由 F1 的子问题的数量乘以 F2 的子问题的数量限制。

为了计算子问题的数目，定义一个有根的，有序森林 F(i,j)-删除的子森林，$0 \le i + j \le |F|$，作为从 F 获得的森林，首先通过重复 j 次地删除最右侧的根，然后，类似地，删除最左侧的根 i 次。我们把(0, j)-删除和(i, 0)-删除称之为子森林，重复 F 的前缀和后缀，区间 0 ≤ j ≤ |F|。

F(i,j)-删除的子森林数量是 $\sum_{k=0}^{|F|} k = O(|F|^2)$，因为对于每个 i，存在|F|-j 的 i 选择。

不难发现，通过引理 1 的递归获得的所有子问题对：S1 和 S2 是 F1 和 F2 的被删除的子森林。

因此，通过上述讨论，时间复杂度受 $O(|F_1|^2|F_2|^2)$ 限制。事实上，子问题可以被需要得更少，我们将会在下面的章节中摆出这个情况。

3.1.1 Zhang 和 Shasha 的算法

以下算法是由 Zhang 和 Shasha[55]提出。定义有序树 T 的关键根如下：

$$\text{keyroots}(T) = \{\text{root}(T)\} \cup \{v \in V(T) \mid v \text{有一个左兄弟}\}$$

T 的特殊子森林是森林 F(v)，其中 v∈keyroots(T)。针对关于 keyroots 的 T 的相关子问题是所有特殊子森林 F(v)的前缀。在本章节中，我们将这些称为相关子问题。

引理 2 对于每个节点$v \in |V(T), F(v)$是一个相关的子问题。

这很容易看出，事实上，可以在上述递归中出现的子问题是形式 F(v)的子森林，其中 v∈V(T)，或 T 的特殊子森林的前缀。因此，遵循引理 2 和相关子问题的定义，为了计算(F1，F2)，分别为 T1 和 T2 的所有相关子问题 S1 和 S2 计算(S1，S2)就足够了。（P9 完）

树 T 的相关子问题可以计数如下。对于节点 v∈V(T),定义 v 的折叠深度 cdepth(v),使其作为 v 的关键根节点祖先的数量，并且将 cdepth(T)定义为所有节点的最大折叠深度 v∈V(T)。

引理 3 对于有序树 T，相对于关键字的相关子问题的数量由$O(|T|\text{cdepth}(T))$限制。

证明。相关子问题可以使用以下表达式计算：

$$\sum_{v \in \text{keyroots}(T)} |F(v)| < \sum_{v \in \text{keyroots}(T)} |T(v)| = \sum_{v \in V(T)} \text{cdepth}(v) \leq |T|\text{cdepth}(T)$$

由于子森林 F(v)的数量前缀是$|F(v)|$，即第一和计数 F（v）的相关子问题的数量。为了证明第一个等式，需要注意对于每个节点 v，包含 v 的特殊子森林的数量是 v 的折叠深度。因此，v 对左侧和右侧贡献相同的量。其他等式/不等式紧随其后。

引理 4 对于一个树 T，$\text{cdepth}(T) \leq \min\{\text{depth}(T), \text{leaves}(T)\}$

因此，使用动态规划可以解决在时间（和空间）上的问题$O(|T_1||T_2| \min\{D_1, L_1\} \min\{D_2, L_2\})$。此外，通过小心地去除特殊森林的前缀的距离，在计算中使用到的空间可以被减少至$O(|T_1||T_2|)$。

所以，**定理 1 （[55]）**对于有序树 T1 和 T2，编辑距离问题可以在时间$O(|T_1||T_2| \min\{D_1, L_1\} \min\{D_2, L_2\})$和空间$O(|T_1||T_2|)$中求解。

3.2.2 Klein 的算法

在最糟糕的情况下，对于具有线性深度和线性叶数的树，上一节的 Zhang 和 Shasha 的算法仍然需要$O(|T_1|^2|T_2|^2)$时间作为简单算法。在[25]Klein 获得了一个比较好的最糟糕情况下

$O(|T_1|^2|T_2|\log|T_2|)$的时间边界，同时也保持$O(|T_1||T_2|)$相同的空间边界。应当注意的是，本文仅将$O(|T_1|^2|T_2|\log|T_2|)$表示为空间界限。然而，这也直接将它改进到$O(|T_1||T_2|)$[23]。该算法基于引理 1 中的递归的扩展。主要思想是考虑 T1 的所有$O(|T_1|^2)$ 个被删除的子森林，但是仅考虑 T2 的$O(|T_2|\log|T_2|)$个被删除的子森林。总而言之，子问题的最糟糕情况的数目因此减少到上述期望界限。

算法中的一个关键概念是将有根的树 T 分解成为重路径的不相交路径。这种技术是由 Harel 和 Tarjan[15]提出的。我们定义节点$v \in V(T)$ as $|T(v)|$.的大小。如下我们将 T 的每个节点分为重的或轻的。根是轻的。对于每个内部节点 v，我们在 v 的子节点中选择 v 的最大大小的子节点 u，并将 u 分类为重的。剩下的子节点都是轻的。我们称一个轻的子节点的边缘为一个轻的边缘，一个重的子节点的边缘为一个重的边缘。节点 v 的轻深度 ldepth(v)，是从 v 到根的路径上的轻边缘的数量。

引理 5 对于任何树 T 和任何$v \in V(T)$, $\text{ldepth}(v) \leq \log|T|+O(1)$。

通过去除轻边缘，T 被分割进入重路径。

我们如下定义了与 T 和轻的子节点有关联的子问题。我们将在本章节中将这些作为相关的子问题。首先修复 T 的重路径分解。对于 T 中的节点 v，我们递归地定义 F(v)的相关子问题如下：F(v)是相关的。如果 v 不是叶，则 u 是 v 的重的子节点，并且 l 和 r 分别是 F(v)中 u 的左边和右边的节点数。然后，F(v)的(i, 0)-删除子森林，$0 \leq i \leq l$，和 F(v)的(l, j)-删除子森林，$0 \leq j \leq r$是相关子问题。递归地，F(u)的所有相关子问题都是相关联的。

关于轻节点的 T 的相关子问题是 F(v)的所有相关子问题的并集，其中 v∈V(T)是轻节点。

引理 6 对于有序树 T，对于轻节点的相关子问题的数量由$O(|T|\,\text{ldepth}(T))$限制。

证明。同理于引理 3 的计算。

还需要注意的是，引理 2 仍然保持相关子问题的这个新定义。令 S 是 T 的相关子问题，并且使 vl 和 vr 分别表示 S 的最左和最右根。如果 S-vr 是相关的，则 S 的差分节点是 vr，如果 S-vl 是相关的，则 s 的差分节点是 v1。引理 1 的递归式是比较最右边的根。显然，我们还可以选择比较最左侧的根使其变成新的递归——我们将其称为引理 1 的对偶。不同的子问题的出现取决于我们使用的递归。我们现在给出一个修改的动态程序来计算树编辑距离。设 S1 是 T1 的删除树，并且使 S2 是 T2 的相关子问题。令 d 是 S2 的差分节点。我们如下计算$\delta(S_1, S_2)$。但有两种情况需要考虑：

1. 如果 d 是 S2 的最右边的根，使用引理 1 比较 S1 和 S2 的最右边的根。
2. 如果 d 是 S2 的最左边的根，比较 S1 和 S2 的最左边的根，使用引理 1 的对偶。

从两种情况中都很容易看出，S1 较小子问题将都是 T1 删除子森林，而而 S2 的较小子问题

将全部是 T2 的相关子问题。使用与 Zhang 和 Shasha 的算法中类似的动态规划技术，我们获得了以下内容：

定理 2 ([25]) 对于有序树 T1 和 T2，编辑距离问题可以在时间$O(|T_1|^2|T_2|\log|T_2|)$和空间$O(|T_1||T_2|)$中求解。

Klein [25] 还表示他的算法可以在与 T1 和 T2 之间的无根有序编辑距离问题相同的时间和空间范围内扩展，定义为在所有 T1 和 T2 可能根上 T1 和 T2 之间的最小编辑距离。

3.3 一般化无序编辑距离

在下一章节中，我们将调查无序编辑距离问题。这个问题显示为 NP-complete[58, 50, 57]，甚至对于具有尺寸 2 的标签字母的二叉树也是如此。减益是从 3-set[12]问题的 Exact Cover 来的。接着，问题显示成 MAX-SNP 困难[54]。所以，除非 P=NP，否则对于这个问题来说没有 PTAS[4]。对于问题的特殊情况[58]，存在多项式时间算法。如果 T2 有一个叶，即 T2 是一个序列，则问题可以在$O(|T_1||T_2|)$时间内解决。更通常的说，存在在时间$O(|T_1||T_2| + L_2!3^{L_2}(L_2^3 + D_1^2)|T_1|)$中运行的算法。因此，如果 T2 中的叶的数量是对数的，则问题可以在多项式时间中求解。因此，如果 T2 中的叶数是对数形式，则问题可以在多项式时间内求解。

3.4 约束性编辑距离

事实上，由于一般的编辑距离问题很难解决，所以导致了对问题的研究版本很有限。在[51,52]中，Zhang 介绍了由δ_c表示的约束编辑距离，其被定义为在不相交子树应当被映射到不相交子树的限制下的编辑距离。形式上，$\delta_c(T_1,T_2)$被定义为满足附加约束的最小成本映射(M_c,T_1,T_2)，对于所有$(v_1,w_1),(v_2,w_2),(v_3,w_3)\in M_c$：

·nca(v1, v2)对于 v3 来说是个合适的祖先，iff nca(w1,w2)对于 w3 来说是个合适的祖先。

根据[29]，Richter [37]独立引入了编辑距离δ_s的结构。类似于约束性编辑距离，$\delta_s(T_1,T_2)$被定义为满足附加约束的最小成本映射(M_s,T_1,T_2)，对于所有$(v_1,w_1),(v_2,w_2),(v_3,w_3)\in M_s$，使得 v1，v2 和 v3 中没有一个是其他的祖先，

. $\mathrm{nca}(v_1,v_2)=\mathrm{nca}(v_1,v_3)$ iff $\mathrm{nca}(w_1,w_2)=\mathrm{nca}(w_1,w_3)$

这直接表明了两个编辑距离的概念是等价的。自此，我们将它们简单地称为约束性编辑距离。作为示例请看图 4 的映射。

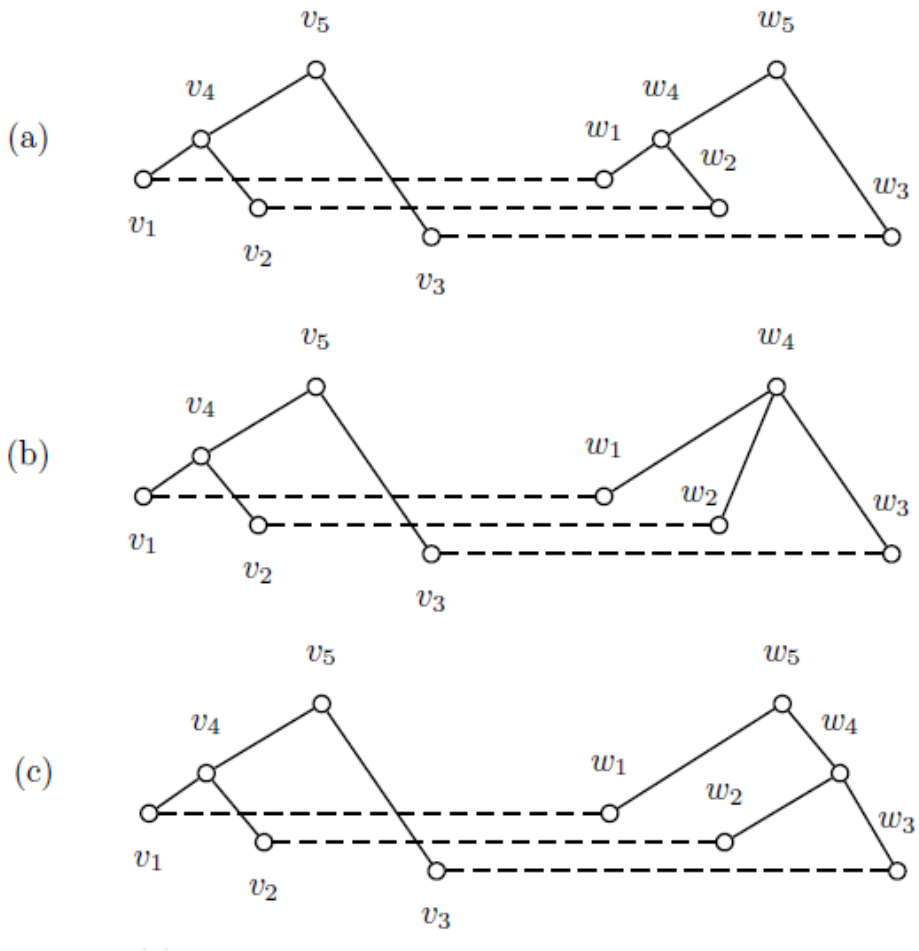

图 4：(a)受约束和约束较少的映射。(b)较少约束但不受约束的映射。(c)既不受约束也不较不受约束的映射。

由于$\mathrm{nca}(v_1, v_2) \neq \mathrm{nca}(v_1, v_3)$ and $\mathrm{nca}(w_1, w_2) \neq \mathrm{nca}(w_1, w_3)$，(a)显示的是一个约束性映射。

（b）没有被约束，由于$\mathrm{nca}(v_1, v_2) = v_4 \neq \mathrm{nca}(v_1, v_3) = v_5$，但是(b) nca(w1,w2) = w4 = nca(w1,w3)。

(c)由于$\mathrm{nca}(v_1, v_3) = v_5 \neq \mathrm{nca}(v_2, v_3)$也没被约束，但(c) $\mathrm{nca}(w_1, w_3) = v_5 \neq \mathrm{nca}(w_2, w_3) = w_4$

在[51,52]中，Zhang 提出计算最低成本约束映射的算法。对于有序的情况，他给出使用$O(|T_1||T_2|)$时间的算法，对于无序的情况，他利用运行时间 O(|T1||T2|(I1+I2)log(I1+I2)).两者都使用空间 O(|T1||T2|).两种算法中的思想概念是相似的。由于对映射的限制，需要考虑较少的子问题，并且获得更快的动态程序。在有序的情况下，主要观察的是减少到字符串编辑距

离问题。对于无序情况，相应的减少是针对于最大匹配问题的。Zhang 利用一个计算最小成本最大流量的有效算法，得到了上述时间复杂度。Richter 提出了一种使用 O(|T1||T2|I1I2)时间和 O(T1|D2I2)空间的有序约束性编辑距离问题的算法。因此，对于小程度，低深度树，该算法与张的算法相比的提供了空间改进。

就在最近，Lu 等人[29]引入了存在较少约束性的编辑距离，δ_l，让约束性映射得到了放宽。这里的要求针对于所有$(v_1, w_1), (v_2, w_2), (v_3, w_3) \in M_l$，使得 v1, v2 和 v3 都不是其他的祖先。

- $\text{depth}(\text{nca}(v_1, v_2)) \geq \text{depth}(\text{nca}(v_1, v_3))$ 和 $\text{nca}(v_1, v_3) = \text{nca}(v_2, v_3)$，当且仅当 $\text{depth}(\text{nca}(w_1, w_2)) \geq \text{depth}(\text{nca}(w_1, w_3))$和 nca(w1,w3)=nca(w2,w3)。

考虑如图 4 中的映射。(a)受约束较小，因为它是约束性的。(b)不是约束性映射，然而映射还是因为 depth(nca(v1, v2)) > depth(nca(v1, v3))，nca(v1, v3) =nca(v2, v3), nca(w1,w2) = nca(w1,w3), 以及 nca(w1,w3) = nca(w2,w3)而受到较小约束。c)不是较少约束的映射，因为 depth(nca(v1, v2)) > depth(nca(v1, v3))和 nca(v1, v3) = nca(v2, v3),，而 nca(w1,w3) 6= nca(w2,w3).

在文献[29]中，提出了使用$O(|T_1||T_2|I_1^3 I_2^3 (I_1+I_2))$时间和空间的较少约束的编辑距离问题的有序版本的算法。对于无序版本，不同于约性编辑距离问题，它的问题是 NP-complete。在这边使用的减益与用于无序编辑距离问题的减益类似。还有报道说，问题是 MAX SNP-hard。此外，对于无序约束的编辑距离问题没有绝对近似算法，除非 P=NP。
（如果存在常数 c>0，则近似算法 A 是绝对的，使得对于每个实例 I，$|A(I) - OPT(I)| \leq c$，其中 A（I）和 OPT（I）分别是 I 的近似和最优解[33]。）

3.5 其他亚类

在本章节中，我们调查编辑距离的其他变体的结果。设 T1 和 T2 为有根树。 T1 和 T2 之间的单位成本编辑距离被定义为将 T1 转变为 T2 所需的编辑操作的数量。在[41]中考虑了这个问题的有序版本，并以此提出了一个快速算法。如果 u 是 T1 和 T2 之间的单位成本编辑距离，则算法在 O(u2 min{|T1|,|T2|}min{L1,L2})时间运行。这个算法使用了 Ukkonen[47]和 Landau 以及 Vishkin[28]的技术。

在[38]中，Selkow 考虑到了插入和删除被限制到树的叶子的编辑距离问题。该编辑距离有时被称为 1-度编辑距离。他给出了一个使用 O(|T1||T2|)时间和空间的简单算法。Lu[30]给出了另一个在子树上编辑操作不是在节点上操作的编辑距离度量。Tanaka 在[45,44]中给出了类似的编辑距离。Lu 的算法的简短描述可以在[42]中找到。

4. 树对齐距离

在本章节中，我们考虑了对齐距离问题。令 T1 和 T2 为根，标记为树，并且使得 γ 作为第 2

章节中定义的标签对上的度量成本函数。通过首先将标记有λ（称为空间）的节点插入到 T1 和 T2 中，使得当标签被忽略时，然后将第一增强树覆盖在另一个上，来获得 T1 和 T2 的对齐 A。A 中的一对相对标签的成本由γ给出。A 的成本是 A 中所有相对标签的成本的总和。T1 和 T2 的最佳对齐是最小成本的 T1 和 T2 的对齐。我们用$\alpha(T_1, T_2)$表示这个成本。图 5 显示了有序对齐的一个例子（从[18]）

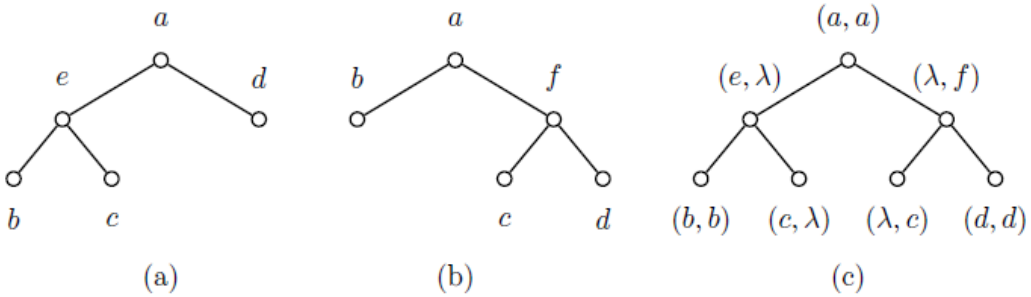

图 5：(a)树 T1。(b)树 T2。(c)T1 和 T2 的对齐。

树对齐距离问题是树编辑问题的特殊情况。实际上，它对应于约束性编辑距离，其中所有的插入必须在任何删除之前执行。所以，$\delta(T_1, T_2) \leq \alpha(T_1, T_2)$。例如，假定所有编辑操作具有成本 1，考虑图 1 中的示例。编辑操作的最佳序列通过删除标记为 e 的节点，然后插入标记为 f 的节点来实现。因此，编辑距离是 2。然而，最佳对准是(c)中描绘的具有值 4 的树。众所周知的事实是，编辑和对准距离在序列的复杂性方面是等效的，在 Gusfield[14]有提到。然而，这对于树来说并不是真实的，具体的我们将在以下章节中显示。在 4.1 章节和 4.2 章节中，我们分别调查了有序和无序树对齐距离问题的结果。

4.1 有序树对齐距离

在本章节中，我们考虑了有序树对齐距离问题。使 T1 和 T2 成为两个有根，有序和标记的树。有序树对齐距离问题是由 Jiang 等人在[18]中介绍的。在那里提出的算法使用 O(|T1||T2|(I1+I2)2)时间和空间。因此，对于小度树，该算法一般会比用于编辑距离的最佳已知算法快。我们在下一章节里会详细介绍这个算法。近来，在[17]中，提出了一种用于类似树的新算法。具体来说，如果使用最多 s 个空间存在 T1 和 T2 的最佳对齐，则算法在时间 O((|T1|+|T2|) log(|T1|+|T2|)(I1+I2)4s2)中计算对齐。该算法以类似于用于比较类似序列的快速算法的方式工作，比如可以参考在第 3.3.4 章节里的[39]。主要概念是通过仅考虑其大小相差最多 O(s)的 T1 和 T2 的子树将 Jiang 等人的算法进行进一步的提速。

4.1.1 Jiang,Wang,和 Zhang 的算法

在本章节中，我们提出了 Jiang 等人的算法[18]。我们只提供如何计算对齐距离。相应的对齐在相同的复杂性边界内可以被容易地构建。使得γ成为标签上的度量成本函数。为简单起见，我们将引用节点而不是标签，也就是说，我们将使用(v，w)来表示节点 v 和 w(label(v), label(w))。这里的 v 或 w 可以是λ。我们将α的定义扩展包括直至森林对齐，即用α(F1, F2)表示森林 F1 和 F2 的最优对齐的成本。

引理 7 使得$v \in V(T_1)$和$w \in V(T_2)$分别与子节点$v_1, \ldots, v_i$和$w_1, \ldots, w_j$。 然后，

$$\begin{aligned}
\alpha(\theta, \theta) &= 0 \\
\alpha(T_1(v), \theta) &= \alpha(F_1(v), \theta) + \gamma(v, \lambda) \\
\alpha(\theta, T_2(w)) &= \alpha(\theta, F_2(w)) + \gamma(\lambda, w) \\
\alpha(F_1(v), \theta) &= \sum_{k=1}^{i} \alpha(T_1(v_k), \theta) \\
\alpha(\theta, F_2(w)) &= \sum_{k=1}^{j} \alpha(\theta, T_2(w_k))
\end{aligned}$$

引理 8 使得$v \in V(T_1)$和$w \in V(T_2)$分别与子节点$v_1, \ldots, v_i$和$w_1, \ldots, w_j$。 然后，

$$\alpha(T_1(v), T_2(w)) = \min \begin{cases} \alpha(F_1(v), F_2(w)) + \gamma(v, w) \\ \alpha(\theta, T_2(w)) + \min_{1 \le r \le j} \{\alpha(T_1(v), T_2(w_r)) - \alpha(\theta, T_2(w_r))\} \\ \alpha(T_1(v), \theta) + \min_{1 \le r \le i} \{\alpha(T_1(v_r), T_2(w)) - \alpha(T_1(v_r), \theta)\} \end{cases}$$

证明。考虑 T1(v)和 T2(w)的一个最佳对齐 A。存在四种情况：(1)(v，w)是 A 中的标签，(2)(v，λ)和(k，w)是 A 中对于某些$k \in V(T_1)$的标签，(v，h)是针对某些$h \in V(T_2)$在 A 中的标签，或者是在 A 中的(4) λ和(λ，w)。由于两个节点可以被删除并且被单个节点(v, w)作为新根，所以不需要考虑情况(4)。由三角不等式所得到的对齐成本至少比较小。

案例 1：A 的根由(v，w)标记。因此，

$$\alpha(T_1(v), T_2(w)) = \alpha(F_1(v), F_2(w)) + \gamma(v, w)$$

案例 2：A 的根由(v, λ)标记。因此，对于有些$1 \le r \le i$来说，$k \in V(T_1(w_s))$。它遵循，

$$\alpha(T_1(v),T_2(w))=\alpha(T_1(v),\theta)+\min_{1\le r\le i}\{\alpha(T_1(v_r),T_2(w))-\alpha(T_1(v_r),\theta)\}$$

案例：对称案例 2。

引理 9 使得$v\in V(T_1)$和$w\in V(T_2)$分别与子节点$v_1,\ldots,v_i$和$w_1,\ldots,w_j$。 对于任何 s，t, 使得$1\le s\le i$和$1\le t\le j$,

$$\alpha(F_1(v_1,v_s),F_2(w_1,w_t))=\min\begin{cases}\alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_{t-1}))+\alpha(T_1(v_s),T_2(w_t))\\ \alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_t))+\alpha(T_1(v_s),\theta)\\ \alpha(F_1(v_1,v_s),F_2(w_1,w_{t-1}))+\alpha(\theta,T_2(w_t))\\ \gamma(\lambda,w_t)+\min\limits_{1\le k<s}\{\alpha(F_1(v_1,v_{k-1}),F_2(w_1,w_{t-1}))\\ \qquad\qquad+\alpha(F_1(v_k,v_s),F_2(w_k))\}\\ \gamma(v_s,\lambda)+\min\limits_{1\le k<t}\{\alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_{k-1}))\\ \qquad\qquad+\alpha(F_1(v_s),F_2(w_k,w_t))\}\end{cases}$$

证明。考虑$F_1(v_1,v_s)$和$F_2(w_1,w_t)$的最佳对齐 A。A 中最右侧树的根标记为(v_s,w_t), (v_s,λ)或(λ,w_t)。

案例 1：标签为(v_s,w_t)。然后 A 最右边的树必须是$T_1(v_s)$和$T_2(w_t)$的最佳对齐。所以，

$$\alpha(F_1(v_1,v_s),F_2(w_1,w_t))=\alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_{t-1}))+\alpha(T_1(v_s),T_2(w_t))$$

案例 2： 标签为(v_s,λ)。然后$T_1(v_s)$与子森林$F_2(w_{t-k+1},w_t)$对齐，其中 0≤k≤t。可能发生以下子过程：

2.1 (k=0)。$T_1(v_s)$与$F_2(w_{t-k+1},w_t)=\theta$对齐。所以，

$$\alpha(F_1(v_1,v_s),F_2(w_1,w_t))=\alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_t))+\alpha(T_1(v_s),\theta)$$

2.2 (k=1)。$T_1(v_s)$与$F_2(w_{t-k+1},w_t)=\theta$对齐。形同案例 1。

2.3 (k ≥ 2)。最常见的案例。可以很容易发现：

$$\alpha(F_1(v_1,v_s),F_2(w_1,w_t)) = \gamma(v_s,\lambda) + \min_{1\le r<t}\{\alpha(F_1(v_1,v_{s-1}),F_2(w_1,w_{k-1})))$$
$$+ \alpha(F_1(v_s),F_2(w_k,w_t)).$$

案例 3： 标签为(λ, w_t)。与案例 2 对称。

该递归式可以用于构造自下而上的动态规划算法，考虑分别具有子$v_1,\ldots,v_i$和$w_1,\ldots,w_j$的固定节点 v 和 w 的对。我们需要为所有 1≤h≤k≤i 计算值$\alpha(F_1(v_h,v_k),F_2(w))$，并且对于所有 1≤h≤k≤j 计算$\alpha(F_1(v),F_2(w_h,w_k))$。也就是说，我们需要计算$F_1(v)$与$F_2(w)$的每个子森林的最佳对齐，另一方面，我们也需要计算$F_2(w)$与$F_1(v)$的每个子森林的最佳对齐。对于任何 s 和 t，1≤s≤i 和 1≤ t≤j，定义集合：

$$A_{s,t} = \{\alpha(F_1(v_s,v_p),F_2(w_t,w_q)) \mid s \le p \le i, t \le q \le j\}$$

为了计算上述的对齐，我们需要对所有 1≤s≤i 和 1≤t≤j 进行$A_{s,1}$和$A_{1,t}$的计算。假设较小子问题的值是已知的，则不难在时间$O((i-s)\cdot(j-t)\cdot(i-s+j-t)) = O(ij(i+j))$中使用引理 9 来计算$A_{s,t}$。因此，计算$A_{s,1}$和$A_{1,t}$（1≤s≤i 和 1≤t≤j）的时间由$O(ij(i+j)^2)$限制。因此，所有节点 v 和 w 所需的总时间由下式限定：

$$\begin{aligned}
&\sum_{v\in V(T_1)}\sum_{w\in V(T_2)} O(\deg(v)\deg(w)(\deg(v)+\deg(w))^2)\\
&\quad\le \sum_{v\in V(T_1)}\sum_{w\in V(T_2)} O(\deg(v)\deg(w)(\deg(T_1)+\deg(T_2))^2)\\
&\quad\le O((I_1+I_2)^2\sum_{v\in V(T_1)}\sum_{w\in V(T_2)}\deg(v)\deg(w))\\
&\quad\le O(|T_1||T_2|(I_1+I_2)^2)
\end{aligned}$$

总之，我们有了以下的定理：

定理 3 ([18]) 对于有序树 T1 和 T2，树对齐距离问题可以在$O(|T_1||T_2|(I_1+I_2)^2)$时

间和空间中求解。

4.2 无序树对齐距离

上面提出的算法可以直接被修改成可以直接处理问题的无序版本的[18]。如果树具有有界度，算法仍然可以在 O(|T1|T2|)时间中运行。如果树具有有界度，这应该被看作与编辑距离问题相反的，即与 MAX SNP-hard 相反的。如果一个树具有任意度，无序对齐即成为 NP-hard [18]。对于编辑距离问题，减益是从减益是从 3-set[12]问题的 Exact Cover 来的。

5. 树包含

在本章节中，我们将调查树包含问题。令 T1 和 T2 成为有根标记树。如果在 T2 上执行删除操作的序列，使得 T1 与 T2 同形，则 T1 包括在 T2 中。树包含问题主要是确定 T1 是否包括在 T2 中。图 6(a)示出了有序包含的示例。树包含问题是树编辑距离问题的特殊情况：如果插入全部具有成本 0，并且所有其他操作具有成本 1，则 T1 可以包括在 T2 中，并且仅当 $\delta(T_1, T_2) = 0$ 被包括在 T2 中。根据[7]树包含问题最初是由 Knuth [26][练习 2.3.2-22]引入的。

本节的其余部分内容如下。在第 5.1 章节中我们给出了一些初步性内容，在第 5.2 和 5.3 章节中，我们分别调查了有序和无序树包含的已知结果。

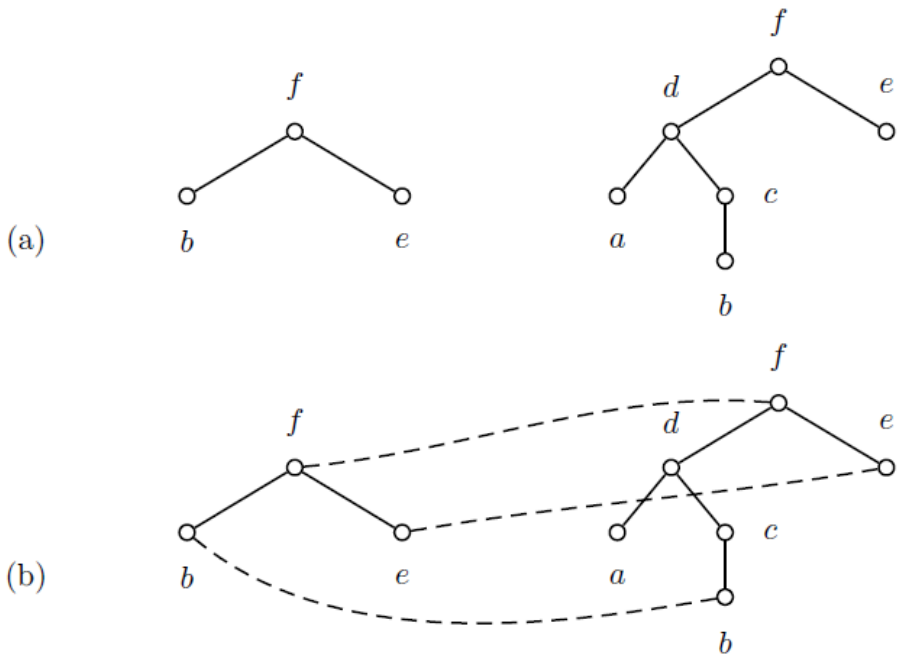

图 6：(a)左侧的树通过删除标记为 d，a 和 c 的节点而包括在右侧的树中。(b)对应于(a)的嵌

入。

5.1 定序和嵌入

让 T 成为一个有标签的，有序的和有根的树。我们定义一个由 $v \prec v' \text{ iff } \text{post}(v) < \text{post}(v')$ 给出的 T 的节点的定序。同样，$v \preceq v' \text{ iff } v \prec v' \text{ or } v = v'$。此外，我们用两个特殊节点$\perp$和$\top$来扩展该排序，使得对于所有节点 $v \in V(T), \perp \prec v \prec \top$。节点 v∈V（T）的左亲属 lr（v）是在 v 左边的节点集合，类似地，右亲属 rr（v）是位于 v 右边的节点集合。

令 T1 和 T2 成为有根标记树。我们定义有序嵌入(f，T1，T2)作为注入函数 f：V(T1)→V(T2)，使得对于所有节点 v，u∈V，

·标签(v) = 标签(f(v))。（标签保存条件）
·v 是 u 的祖先 iff f(v)是 f(u)的祖先。（祖先条件）
• v 在 u 的左边 iff f(v)在 f(u)的左边。（同胞条件）

因此，嵌入是映射的特殊情况（参考第 3.1 章节）。无序嵌入如上定义，但没有同胞条件。嵌入(f，T1，T2)是 iff(root(T1))=root(T2)的根保留。图 6（b）示出了根保留嵌入的示例。

5.2 有序树包含

让 T1 和 T2 成为有根，有序和标记的树。有序树包含问题已经引起了许多研究的关注。Kilpeläinen和 Mannila (同样参考 [21]) 提出了使用 O（|T1||T2|）时间和空间的第一个多项式时间算法。大多数后来的改进对该算法进行的细化。在下一章节我们会详细介绍这个算法。在[21]中，使用 O(|T1|D2)空间给出了上述的空间效率更高的版本。 Richter 给出了使用 $O(|\Sigma_{T_1}||T_2| + m_{T_1,T_2}D_2)$时间的算法，其中 Σ_{T_1} 是 T1 的标签的字母表，m_{T_1,T_2} 是设置的匹配，定义为对(v，w) ∈ T1×T2 的数量，使得 label(v)=label(w)。

5.2.1 Kilpeläinen和 Mannila 的算法

在本章节中，我们为有序树包含问题提出了Kilpeläinen和 Mannila [22]的算法。让 T1 和 T2 成为有序标记树。定义 R(T1，T2)作为 T1 的根保留嵌入到 T2。我们定义 $\rho(v, w)$，其中 v∈V(T1)和 w∈V(T2)：

$$\rho(v, w) = \min_{\prec} (\{w' \in rr(w) \mid \exists f \in R(T_1(v), T_2(w'))\} \cup \{\top\})$$

所以，$\rho(v,w)$是具有 T1(v)根保留嵌入的最接近 w 的右相对。此外，如果不存在这样的嵌入，则$\rho(v,w)$是$\top$。根据定义，我们容易发现，T1 可以被包括在 T2 中，并且仅当$\rho(v,\perp) \neq \top$。以下引理指明了如何搜索根保留嵌入。

引理 10 令 v 成为 T1 中的一个与子 v1, . . . , vi 同在的节点。对于一个在 T2 内的节点 w，通过设置$p_1 = \rho(v_1, \max_{\prec} \mathrm{lr}(w))$和$p_k = \rho(v_k, p_{k-1})$, 2 ≤ k ≤ i，来定义序列 p1, . . . , pi。当且仅当 label(v)= label(w)和 pi ∈ T2(w)时，对于所有的 1≤k≤i，存在 T2(v)中 T1(v)的根保留嵌入 f。

证明。如果在 T1(v)和 T2(w)之间存在根保留嵌入，则直接检查是否存在序列 pi，1≤i≤k，使其能够满足条件。相反，假设$p_k \in T_2(w)$对于所有 1≤k≤，并且 label(v)=label(w)。我们如下构造 T1(v)的根保留并嵌入 f 到 T2(w)中。令 f(v)=w。根据ρ的定义，在$T_1(v_k)$中必须有$f^k, 1 \leq k \leq i$的根保留嵌入$T_2(p_k)$，1≤k≤i。对于$T_1(v_k)$中的节点 u，1≤k≤i，我们设置$f(u) = f^k(u)$。由于所有 k 的$p_k \in \mathrm{rr}(p_{k-1}), 2 \leq k \leq i$和$p_k \in T_2(w)$，1≤k≤i，因此 f 确实是根保留嵌入。

使用动态规划，现在可以直接计算所有$v \in V(T_1)$和$w \in V(T_2)$的$\rho(v,w)$。对于固定节点 v，我们以反向后序遍历 T2。在每个节点 w ∈ V(T2)，我们检查 T1(v)在 T2(w)中是否存在根保留嵌入。如果实这样的话，我们对于所有$q \in lr(w)$设置$\rho(v,q) = w$，使得$x \preceq q$，其中 x 是 T1(v)在 T2(w)中的下一个保留根的嵌入。

对于一对节点 v ∈ V(T1)和 w ∈ V(T2)，我们使用引理 10 来测试根保留嵌入。假设已经计算了较小子问题的值，所使用的时间是 O(deg(v))。因此，对节点 w 的总时间贡献为$\sum_{v \in V(T_1)} O(\deg(v)) = O(|T_1|)$。因此，算法的时间复杂度受 O(|T1||T2|)的限制。显然，只需要 O(|T1||T2|)空间来存储ρ。因此，我们得到了以下定理，

定理 4 ([22]) 对于任何一对有根，有标记和有序的树 T1 和 T2，树包含问题可以在 O(|T1||T2|)时间和空间中求解。

5.3 无序树包含

在[22]中，无序树包含问题是 NP-complete。被使用的减益来自满足性问题[12]。Ma toušek和 Thomas[32]独立地提供了 NP-completeness 的另一个证明。使用 O(|T1|I122I1|T2|)时间，在[22]中给出了无序树包含问题的算法。因此，如果 I1 恒定，则算法在 O(|T1||T2|)时间中运行，并且如果 I1=log|T2|该算法在 O(|T1|log|T2||T2|3)中运行

6. 总结

在这篇文章中，我们调查研究了树编辑距离，对齐距离和包含问题。 除此之外，我们也已经以我们自己的观点向大家展示了每个问题的中心算法。 有几个开放性质的问题可能会是接下来进一步研究的主题。 我们在结论中用一个简短的列表提出了一些可行的方向。

• 对于上述问题的无序版本，一些是 NP-complete，其他不是。肯定的是，确定哪些类型的映射，为无序版本提供 NP-complete 问题会提高对所有上述问题的理解。

• 当前最好的有序树编辑距离问题的最糟糕情况的上界限是使用 O(|T1|2|T2|log|T2|)的算法[25]。相反，最长公共子序列问题[1]的二次下界限是有序树编辑距离问题的最佳常规下界限。因此，在复杂性内的大差距需要被关闭。

•取决于具体应用，可以考虑除了上述之外的几个有意义的编辑操作。每组操作会产生一个新的编辑距离问题可以使我们确定复杂性。我们也已经考虑了树编辑距离问题的一些扩展[6,5，24]。

致谢：

感谢 Inge LiGørtz 和 Anna Ostlin 两位的证实性阅读和具有帮助性的讨论。