

國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

一個相容超樹問題的改進演算法

**An Improved Algorithm for
Compatible Supertree Problem**



鄭智中

Chih-Chung Cheng

指導教授：呂學一 博士

Advisor: Hsueh-I Lu, Ph.D.

中華民國98年7月

July, 2009

誌 謝

在完成論文的過程裡，我成長了許多。一方面在論文寫作上有所進步：我學著捕捉難以說明的概念與細節、組織論文讓它展現應有的結構；一方面也培養完成困難事情的能力。

在這樣的過程中，我要感謝我的指導教授呂學一老師。老師對於學問認真與正直的態度，以及在討論上的無盡耐心是最好的身教。在研究的過程中，也提供了極有洞見的建議。這些建議往往能夠讓我從固有的區域最小值中解放出來，在思考上有著更進一步的提升。

接著要感謝我的口試委員，高明達教授與何錦文教授，兩位教授在百忙之中抽空前來參加我的口試，提出了許多寶貴的意見。

我也要感謝實驗室可愛的同學們。每次到實驗室都有著愉快的氣氛，也常有熱烈而即席的討論。演算法的領域很廣，自己研究的範圍很小，常常在討論中了解那些令人驚嘆的美妙結果與複雜符號背後的直覺觀點。尤其要感謝在最後階段一同奮戰的同伴們，偉揚、慶徵、彥朋、與佳慶。一個人做起來很辛苦的事情，和大家一起互相打氣，共同努力，反而會有種革命情感。

接著要感謝我的朋友們，許多朋友以自己的經驗給我建議，像是品光、彥琳、千婷、與坤維。坤維在我最想放棄的時候，以過來人的身份給我最恰當的鼓勵。哲民、燈、彥伯、惟尹、立行、宜家、佳姘、葉怡玉教授與實驗室的同學們、建中輔導室的老師們、瑞呈學長、俊英學長、斯韡學長、傑祥學長都給我不斷的聲援與實質的幫助。許多朋友們也不斷的用各自的方式給我鼓勵，這些鼓勵或大或小，累積起來把事情往好的方向推進。

最後，我要感謝我的家人，爸媽、姊姊、外婆、與二姑姑，你們容忍我各種任性的舉動，給我物資上極大的幫助，最難能可貴的，一個可以放鬆的家。

摘要

在演化生物學中，尋找超樹是個重要的主題。一棵半標籤樹 T 是一個有以下特性的樹：(1) 每個 T 的頂點可以有零到多個標籤；(2) 每個標籤在 T 裡只出現一次；(3) 所有的樹葉與無分叉的頂點都至少有一個標籤。如果滿足以下性質，則稱半標籤樹 T 遠祖顯示另一棵半標籤樹 T' ：(1) 如果 x 在 T' 裡是 y 的祖先，則 x 在 T 裡仍是 y 的祖先；(2) 各自來說，如果在 T' 之中， x, y 不在從樹根到 y, x 的路徑上，則各自來說 x, y 在 T 之中不會在從樹根到 y, x 的路徑上。一個排名半標籤樹 T 是一棵在每個頂點上都有一個排名值的半標籤樹。如果 u 是 v 的祖先，則 u 的排名會比 v 的排名小。如果滿足以下條件，則稱排名半標籤樹 T 保存一個相對分歧時刻 $div(L) < div(L') : lca(L)$ 的排名小於 $lca(L')$ 的排名。其中 L 及 L' 是兩個標籤集合， $lca(L)$ 則是那些標籤所在頂點的最接近共同祖先。

我們提出一個在 $O(h \log^2 h)$ 時間與 $O(h)$ 空間內找到一個排名半標籤樹的演算法，此樹遠祖顯示輸入的一組半標籤樹 P 與一組相對分歧時刻 D 。其中 h 是 P 中每棵樹的標籤數目總和加上 D 中每個相對分歧時刻中的標籤數目總和。

Abstract

Finding supertrees is critical in evolutionary biology. A semi-labeled tree \mathcal{T} is a rooted tree satisfies the following: (1) each node in \mathcal{T} has zero to multiple labels, (2) each label in \mathcal{T} appears only once, and (3) all leaves and non-branching internal nodes have at least one label. A semi-labeled tree \mathcal{T} ancestrally displays another semi-labeled tree \mathcal{T}' if the following are satisfied: (1) if x is an ancestor of y in \mathcal{T}' , then x is an ancestor of y in \mathcal{T} ; and (2) if x, y are not on the path from root to y, x in \mathcal{T}' , respectively, then x, y are not on the path from root to y, x , respectively. A ranked semi-labeled tree \mathcal{T} is a semi-labeled tree with one rank on each node, and if u is an ancestor of v , then the rank of u is smaller than the rank of v . A ranked semi-labeled tree \mathcal{T} preserves a relative divergence date $div(L) < div(L')$ where L and L' are two set of labels, if the rank of $lca(L)$ is smaller than the rank of $lca(L')$ where $lca(L)$ is the least common ancestor of the vertices which the labels $l \in L$ is on. We show a $O(m \log^2 m)$ -time and $O(m)$ -space algorithm to find a ranked semi-labeled tree which ancestrally displays a set P of semi-labeled trees and preserves a set D of relative divergence dates, where m is the sum of the number of labels in each trees in P and of labels in L and L' in each relative divergence dates in D .

Contents

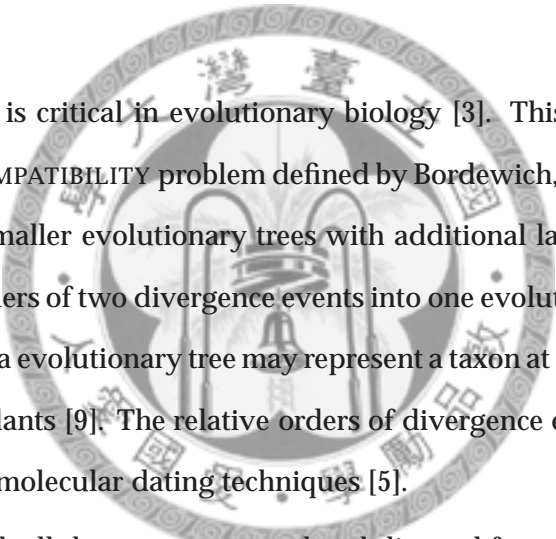
Acknowledgements	i
Chinese Abstract	ii
English Abstract	iii
1 Introduction	1
1.1 Problem definition	2
1.2 Related work	4
1.3 Roadmap	6
2 Preliminary	7
2.1 Generalization of input tree	7
2.2 Decremental connectivity problem	8
2.3 The batch deletion problem	8
3 The BUILDPLUS Algorithm	10
3.1 Terms	10



3.2	A Reduction	10
3.2.1	Constraint Graph	10
3.2.2	The constraint graph compatibility problem	11
3.2.3	The Reduction	12
3.3	The Algorithm	13
3.3.1	Terms	13
3.3.2	Ranked Hierarchy	13
3.3.3	GraphBuild Algorithm	14
3.3.4	The creation of \mathcal{G}_m	15
3.3.5	The BuildPlus algorithm	15
4	Restricted constraint graph	16
4.1	Intermediate vertices	16
4.2	Bundles	17
4.2.1	Red bundles	17
4.2.2	Aqua bundles	23
4.2.3	Replacing the aqua edges with aqua bundles	23
4.3	Proof	26
5	Proof of Theorem 1.1	28
	Bibliography	29

Chapter 1

Introduction



Finding supertrees is critical in evolutionary biology [3]. This paper focuses on the following TREE COMPATIBILITY problem defined by Bordewich, Evans, and Semple [4] which integrates smaller evolutionary trees with additional labels on internal nodes and the relative orders of two divergence events into one evolutionary tree. A label on an internal node of a evolutionary tree may represent a taxon at higher taxonomic level than all its descendants [9]. The relative orders of divergence events can be collected from fossil data or molecular dating techniques [5].

Unless specified, all the trees are rooted and directed from root to leaves, and we use a vertex and the labels on the vertex interchangeably. The label set of a structure S , denoted $L(S)$, is the set of all labels in it.

For any set S , let $|S|$ denote the cardinality of S .

1.1 Problem definition

Let T be a rooted tree. Each node v of T is associated with a set $L_T(v)$ of labels, where $L_T(v)$ is allowed to be empty. Let L_T denote the union of $L_T(v)$ over all nodes v of T . We say that T is a *semi-labeled tree* if both of the following conditions hold.

- *Condition C1*: $L_T(u) \cap L_T(v) = \emptyset$ holds for any two distinct nodes u and v of T .
- *Condition C2*: $L_T(u)$ is not empty for each node u that has at most one child in T .

By Condition C1, we can specify a node u with nonempty $L_T(u)$ by any label in $L_T(u)$. For any two distinct nodes u and v of T , we say that u is an *ancestor* of v if u belongs to the path of T between v and the root of T . For any semi-labeled tree T , define

$$\begin{aligned} D(T) &= \{(a, b) \mid a, b \in L_T, a \text{ is an ancestor of } b \text{ in } T\}; \\ N(T) &= \{\{a, b\} \mid a, b \in L_T, a \text{ and } b \text{ are unrelated in } T\}. \end{aligned}$$

For any two semi-labeled trees T and T' , we say that T *ancestrally displays* T' if $D(T') \subseteq D(T)$ and $N(T') \subseteq N(T)$ [8]. A function r from the nodes of T to non-negative integers is a *rank* of T if $r(u) < r(v)$ holds for any nodes u and v such that u is an ancestor of v in T [5]. A *relative divergence date* on T is a constraint of in the form $div(L) < div(L')$, where L and L' are nonempty subsets of L_T .

A tree T with rank function r *preserves* a relative divergence date $div(L) < div(L')$ if

$$r(lca_T(L)) < r(lca_T(L')).$$

Given a set P of rooted semi-labeled tree and a set D of relative divergence dates, the TREE COMPATIBILITY problem, defined in [4] seeks a ranked semi-labeled tree

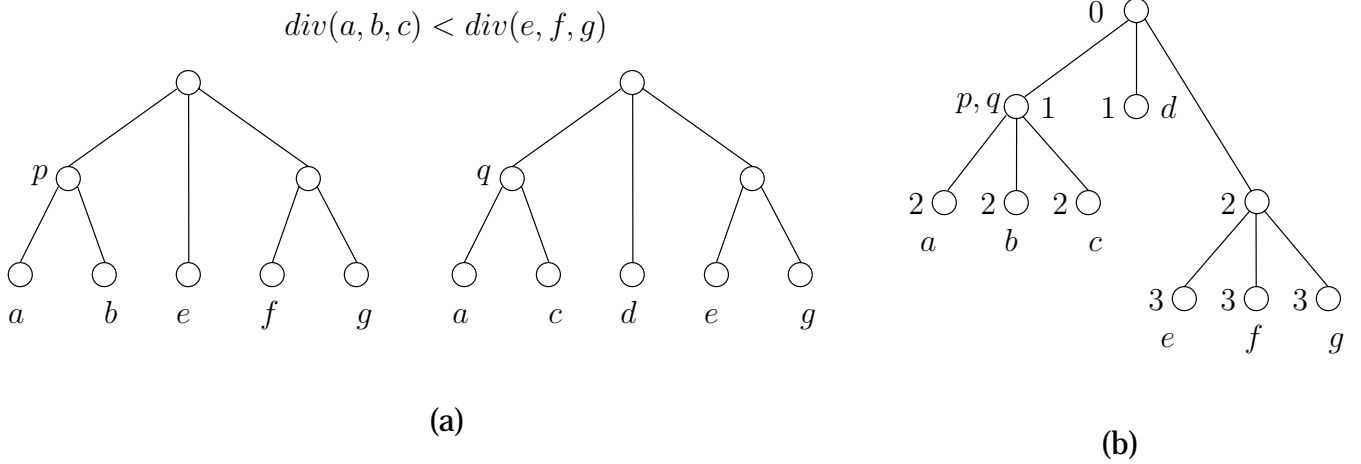


Figure 1.1: A ranked semi-labeled tree in (b) ancestrally displays rooted semi-labeled trees and preserves relative divergence dates in (a).

which ancestrally displays P and preserves D , or reports “incompatible” if no such tree exists.

The ranked semi-labeled tree in figure 1.1b ancestrally displays all rooted semi-labeled trees and preserves all relative divergence dates in figure 1.1a.

The best result is $O(n^3m^3)$ time and $O(n^2m^2)$ space with $n = |L(P) \cup L(D)|$, and $m = |P| + |D|$ by Bordewich, Evans, and Semple [4]. And for each relative divergence dates in the form $div(S) < div(S')$, the size of S and S' are limited to 2 in their algorithm. Moreover, all compatible trees can be enumerated in the same work. [4]

In this work, we reduce the time and space complexity of TREE COMPATIBILITY with a constraint graph based algorithm in the following time and space bound.

Theorem 1.1. *Given a set P of rooted semi-labeled trees and a set D of relative divergence dates, a ranked semi-labeled tree (T, r) which ancestrally displays P and preserves D can be built in $O(h \log^2 h)$ time with $O(h)$ space if such tree exists, or the incompatibility can be*

detected in the same bound if not. And $h = \|P\| + \|D\|$ where $\|P\| = \sum |L(T)|$ for $T \in P$ and $\|D\| = \sum |L(r)|$ for $r \in D$.

The previous best result is $O(n^3m^3)$ time and $O(n^2m^2)$ space, by applying the techniques in [2], it is not hard to acquire a $O(n^2m^2 \log^2 nm)$ time and $O(n^2m^2)$ space algorithm with $n = |L(P) \cup L(D)|$, and $m = |P| + |D|$. Our work further reduce the time to $O(nm \log^2 nm)$ and space to $O(nm)$. While in our notations, the previous best result becomes $O(h^2 \log^2 h)$ time and $O(h^2)$ space with $h = \|P\| + \|D\|$.

Moreover, our result has the following additional features.

1. The ordering property. An algorithm satisfies *ordering property* if with the permutation of input trees as new input, the output tree will be the same one. [20]
2. The renaming property. An algorithm satisfies *renaming property* if with renamed labels as new input, the output tree will be the same one with labels renamed accordingly. [20]
3. Optional compact form of relative divergence date. The relative divergence date in the form $div(S, x) < div(S', x')$ with $x, x' \in \{0, 1\}$ represents all relative divergence dates in the form $div(s) < div(s')$ for any subset $s \in S$ and $s' \in S'$ with $s = \max\{x|S|, 1\}$ and $s' = \max\{x'|S'|, 1\}$.

1.2 Related work

Descending from BUILD. A stream of long developing supertree methods started from applying BUILD algorithm [1] in this domain [6, 7, 14, 18, 19]. The input rooted semi-labeled trees are labeled only on leaves and singularly-labeled. A tree is *singularly-labeled* if each node has at most one label. The BUILD algorithm uses $O(Nn)$ time, and

Henzinger [12] cut its complexity down to $\min\{O(Nn^{1/2}), O(N + n^2 \log n)\}$ with N the number of vertices and n the number of distinct labels. The MINCUTSUPERTREE [17] algorithm and its modified versions [15] improved BUILD to solve incompatible data set with weights on each tree.

In successive works, the input was then expanded in two ways: one is the inclusion of non-tree constraints, and the other is allowing labels for internal vertices.

The inclusion of relative divergence dates. In RANKEDTREE algorithm [5], the input is a set D of relative divergence dates. It outputs a rooted semi-labeled tree with rank. The input rooted semi-labeled trees are labeled only on leaves and singularly-labeled. It runs in $O(|D| + n^3)$ time where $n = |L(D)|$.

The inclusion of rooted semi-labeled trees. In ANCESTRALBUILD algorithm [8], the input is a set of rooted semi-labeled trees. And the output tree ancestrally displays all the trees in input. It runs in $O(t^2n^3)$ where t is the number of trees and n is the number of taxa [2]. When the input trees are all singularly-labeled, the ANCESTRAL-BUILD* algorithm [2] cut down the complexity to $O(\log^2 n \cdot (\sum_{T_i \in P} \sum_{u \in I(T_i)} d(u)^2))$ for $n = |L(P)|$ and $I(T)$ is the set of all internal nodes in tree T . For other variants of ANCESTRALBUILD with singularly-labeled trees, NESTEDSUPERTREE [9] solves the rest types of incompatible inputs, after excluding pairwise inconsistent and ancestor-descendant contradiction. The input P is *pairwise inconsistent* if there exists two labels $x, y \in L(P)$, x is a strict ancestor of y in a tree but x is not strict ancestor of y in some tree T with $\{x, y\} \subseteq L(T)$. P is *ancestor-descendant contradiction* if there is a directed loop in G where $V(G) = L(P)$ and $E(G) = \{(x, y) : x \in L(u), y \in L(v), (u, v) \in T, T \in P\}$.

And MINEDGEWEIGHTTREE [9], a variant of NESTEDSUPERTREE, allows weighted

trees and can be modified easily to have flexible weighting functions on either edges or labels.

The applications. For applications, ANCESTRALBUILD, its variants, and RANKEDTREE are demonstrated on real data or used in [11, 16].

The approaches of RANKEDTREE and ANCESTRALBUILD are combined in the TREE COMPATIBILITY problem [4]. It takes a set P of ranked semi-labeled trees and a set D of relative divergence dates as input, and then rejects incompatible input, or output a ranked semi-labeled tree if input is compatible. The algorithms BUILDPLUS [4] solve this problem in cubic time $O((nm)^3)$ where n is the number of labels in input and m is the number of trees plus the number of relative divergence dates [4]. ALLBUILDPLUS [4] in the same work output all compatible trees.

1.3 Roadmap

The TREE COMPATIBILITY problem is reduced to the CONSTRAINT GRAPH COMPATIBILITY problem in Section 2, the CONSTRAINT GRAPH COMPATIBILITY problem is solved in Section 3, and our result is proved in Section 4.

Chapter 2

Preliminary

The TREE COMPATIBILITY and CONSTRAINT GRAPH COMPATIBILITY problem are abbreviated to \mathcal{TC} and \mathcal{GC} , respectively.

For a graph G , let $C(G)$ be the connected components of G . For a connected component $C \in C(G)$, if, after the removal of a set of vertices in C , C becomes a set of connected components \mathcal{C} , let \mathcal{C} be the *children* of C , denoted $child(C)$. And let the *main* connected component, denoted $M(\mathcal{C})$, be an arbitrarily chosen connected component $C' \in \mathcal{C}$ with $|E(C')| \geq |E(C'')|$ for $C'' \in \mathcal{C}$ and the *new* connected component, denoted $N(\mathcal{C})$, be $\mathcal{C} \setminus M(\mathcal{C})$ where $\mathcal{C} = child(C)$.

2.1 Generalization of input tree

A ranked semi-labeled tree is *fully-labeled* if each node has at least one label. A fully-labeled semi-labeled tree is called *fully-labeled tree*.

Lemma 2.1. *Let P and D be the input of \mathcal{TC} . Without loss of generality, P can be assumed as fully-labeled and with a common root $\rho \notin L(D)$.*

2.2 Decremental connectivity problem

Let $G = (V, E)$ be an undirected graph. The DECREMENTAL CONNECTIVITY PROBLEM maintains G with the operation of edge removal, and the query of whether two vertices are connected.

Lemma 2.2 (Holm et al. [13], Thorup [21]). *The DECREMENTAL CONNECTIVITY PROBLEM can be solved in $O(m \log^2 n + p \log n / \log \log n)$ and $O(m)$ space with p queries, where $n = |V|, m = |E|$.*

2.3 The batch deletion problem

For a graph G supporting the operation of vertex set removal with all vertices removed are in the same component, let W be a *component graph* data structure, if W is a data structure with respect to G maintaining a property of G , and after the removal of a vertex set, G becomes G' , the property of G remains maintained if the update function $U(x)$ of W are called for each $x \in C$, some $C \in \mathcal{C}(G)$.

W has *partial update property* if

1. W is initially maintained.
2. W is maintained, if, after a connected component $C \in \mathcal{C}(G)$ becomes its children \mathcal{C} by removal of some vertices S in $V(C)$, the following is satisfied,
3. For each connected component $C' \in \mathcal{C} \cup S$ except an arbitrary one $C'' \in \mathcal{C}$, C' is updated by the following:
4. The update function provided by W is called for all vertices in C' .

Lemma 2.3 (Even et al. [10], Henzinger et al. [12], and Berry et al. [2]). *For a component graph data structure W with respect to G and update function $U(x)$, and W has the partial update property, if the DECREMENTAL CONNECTIVITY PROBLEM can be solved in $T(n, m, p)$ time and $S(n, m, p)$ space with G and p queries, then W is maintained by calling $U(x)$ at most $O(\log n)$ times for each $x \in V(G)$ with total $O(m \log n + T(n, m, m))$ time and $O(m + S(n, m, m))$ space where n, m are the number of vertices and edges in G , respectively.*



Chapter 3

The BUILDPLUS Algorithm

Since our improvement is based on the BUILDPLUS algorithm [4], we describe it first.

3.1 Terms

For a vertex x and a structure G , let $\text{indegree}_G(x)$ and $\text{outdegree}_G(x)$, denote the indegree and outdegree of x in G , respectively.

3.2 A Reduction

3.2.1 Constraint Graph

A constraint graph $\mathcal{G} = (G, \phi, Q, R)$ consists of

1. a directed graph G ,
2. a coloring function $\phi : V(G) \rightarrow \{\text{blue}, \text{aqua}\}$,
3. a directed graph Q , and

4. a directed graph R ,

where $L(G), V_a(G)$ are the set of blue and aqua vertices in G , respectively.

Let $L(Q) = L(G) \cap V(Q)$, $L(R) = L(G) \cap V(R)$, and $V_a(R) = V_a(G) \cap V(R)$.

In G , for an aqua vertex $a \in V_a(G)$, $\text{indegree}_G(a) = 0$.

In Q , $V(Q) = L(Q) = L(G)$, and $E(Q)$ are called *red edges*.

In R , $V(R) = L(G) \cup V_a(G)$. For a blue vertex $b \in L(R)$, $\text{indegree}_R(b) = 0$. For $a \in V_a(R)$, $\text{outdegree}_R(a) = 0$.

Remark 3.1. The edges in graphs G, Q , and R are constraints. A vertex with indegree 0 in all graphs can be removed in the iteration of BUILDPLUS, and the removed vertices becomes node in output tree.

3.2.2 The constraint graph compatibility problem

A ranked semi-labeled tree \mathcal{T} displays a constraint graph $\mathcal{G} = (G, \phi, Q, R)$, if

1. $L(\mathcal{T}) = L(G)$.
2. For two labels x, y with a directed path from x to y in G , x is an ancestor of y in \mathcal{T} .
3. For an edge $(x, y) \in L(Q)$, x, y are not on the path from root to y, x in \mathcal{T} , respectively.
4. For an aqua vertex a , let $L = \{x \in L(R) : (x, a) \in E(R)\}$, and $L' = \{y \in G : (a, y) \in E(G)\}$. We have $\text{rank}(\text{lca}(L)) < \text{rank}(\text{lca}(L'))$ in \mathcal{T} .

The CONSTRAINT GRAPH COMPATIBILITY problem, \mathcal{GC} for short, try to find a ranked semi-labeled tree \mathcal{T} displays the input constraint graph $\mathcal{G} = (G, \phi, Q, R)$, or

output “incompatible” if not, where all $v \in V(G)$ are all in the same core.

3.2.3 The Reduction

Lemma 3.2. [Bordewich et al. [4]] *The TREE COMPATIBILITY problem can be reduced to CONSTRAINT GRAPH COMPATIBILITY problem. Given an instance of CONSTRAINT GRAPH COMPATIBILITY problem, a constraint graph \mathcal{G} which is reduced from P and D , the instance of TREE COMPATIBILITY, if CONSTRAINT GRAPH COMPATIBILITY returns a ranked hierarchy \mathcal{H} , then \mathcal{H} ancestrally displays P and preserves D .*

The reduction is described as following:

1. Let $L(G) = L(Q) = L(R) = L(P) \cup L(D)$.
2. For an internal node u in T and all its children $v \in \text{child}(u)$, add arcs $\{(x, y) : x \in L(u), y \in L(v)\}$ in G , where $T \in P$.
3. For each pair of siblings $N \in T$, add arcs $E = \{(x, y) : x \in u, y \in v, u \neq v, u, v \in N\}$ in Q .
4. For each relative divergence date $\text{div}(L) < \text{div}(L')$, an aqua vertex a are built in G and R , arcs $\{(a, y) : y \in L'\}$ are built in G , and arcs $\{(x, a) : x \in L\}$ are built in R .
5. For each relative divergence date $\text{div}(L, x) < \text{div}(L', x)$ in the compact form, it is expressed by corresponding relative divergence dates D_i , and then D_i is built by the previous step.

The above version of constraint graph applies the method of restricted descendency graph shown in [2], and by Proposition 4 in [2], it can replace the original ver-

sion of constraint graph in [4].

To prove Lemma 3.2, it is sufficient to show if there exists a ranked semi-labeled tree \mathcal{T} which ancestrally displays P and preserves D , then CONSTRAINT GRAPH COMPATIBILITY returns \mathcal{T} displaying \mathcal{G} . The sufficiency is shown in [4].

Furthermore, if there exists no tree which ancestrally displays P and preserves D , then the CONSTRAINT GRAPH COMPATIBILITY returns “incompatible.”

3.3 The Algorithm

3.3.1 Terms

Let $\mathcal{G} = (G, \phi, Q, R)$ be a constraint graph. A *core* of \mathcal{G} is a connected component in G , and let $C(\mathcal{G})$ denote the cores in G . A *source* of \mathcal{G} is a blue vertex with indegree 0 in G , Q . Let $x \leq_{\mathcal{T}} y$ if x is an ancestor of y . Let $\|\mathcal{G}\|$ be $|V(G)| + |E(G)| + |V(Q)| + |E(Q)| + |V(R)| + |E(R)|$.

3.3.2 Ranked Hierarchy

In [4], the ranked hierarchy is used as a data structure for recording the output tree in the main iteration. A ranked hierarchy $H_{\mathcal{T}}$ can be transformed into a unique ranked semi-labeled tree \mathcal{T} .

A *cluster* is a set of labels. In the corresponding tree \mathcal{T} and a vertex u , the cluster $C_{\mathcal{T}}(u)$ is

$$\bigcup_{v:u \leq_{\mathcal{T}} v} L(v).$$

A *ranked hierarchy* is set of clusters and a rank on each cluster, for each pair of clusters $C_{\mathcal{T}}(u)$ and $C_{\mathcal{T}}(v)$ with $C_{\mathcal{T}}(v) \subseteq C_{\mathcal{T}}(u)$, we have $rank(C_{\mathcal{T}}(u)) < rank(C_{\mathcal{T}}(v))$.

3.3.3 GraphBuild Algorithm

The GRAPHBUILD algorithm to solve CONSTRAINT GRAPH COMPATIBILITY problem. It can be derived from the BUILDPLUS in [4].

Algorithm 1 GraphBuild(\mathcal{G})

- 1: Create \mathcal{G}_0 from \mathcal{G} .
 - 2: **repeat**
 - 3: Iteration k , starting from 0.
 - 4: // **Phase 2**
 - 5: Let $S_a = \{a : a \in V(G_{2k}), a \notin V(Q_{2k})\}$.
 - 6: $\mathcal{G}_{2k+1} \leftarrow \mathcal{G}_{2k} \setminus S_a$.
 - 7: // **Phase 1**
 - 8: Let $S = \{s : \text{indegree}(s) = 0 \text{ in } G_{2k+1}, Q_{2k+1}\} \setminus \{s : \text{indegree}(s) \neq 0 \text{ in } Q_{2k}\}$.
 - 9: $\mathcal{G}_{2k+2} \leftarrow \mathcal{G}_{2k+1} \setminus S$
 - 10: Let π_{k+1} be the partition of the vertex set of G_{2k+2} induced by the cores of G_{2k+2} .
 - 11: $\mathcal{H}_{k+1} \leftarrow \mathcal{H}_k \cup \pi_{k+1}$.
 - 12: **for all** cluster $B \in \pi_k \setminus \pi_{k+1}$ **do**
 - 13: $r'(B) \leftarrow k$.
 - 14: **end for**
 - 15: **until** (a) All vertices in G_{2k+2} are removed, or (b) it is impossible to do so.
 - 16: **Incompatible.** The input is incompatible in condition (b)
 - 17: **Compatible.** In condition (a), \mathcal{H}_k is a ranked semi-labeled tree which ancestrally displays P' and P and preserves D .
 - 18: **Restriction.** Restrict \mathcal{H}_k to $L(P) \cup L(D)$ and get \mathcal{H} .
-

3.3.4 The creation of \mathcal{G}_m

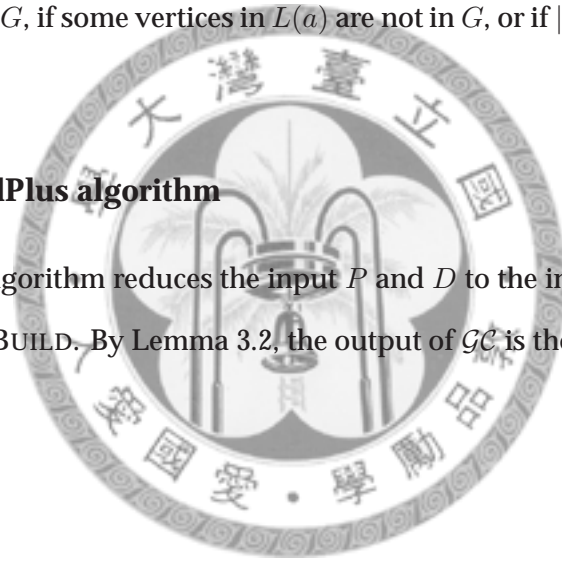
If $|C(\mathcal{G})| = 1$, Let $\mathcal{G}_0 = \mathcal{G}$. Otherwise, create a blue vertex ρ in G, Q, R , and arcs $(\rho, x) \in E(G)$ for each blue vertex $x \in L(G)$.

When the constraint graph \mathcal{G}_{m-1} becomes \mathcal{G}_m , the following maintenance in Q_m is done:

1. The arcs (x, y) with x, y are in different cores are removed.
2. For an aqua vertex a , let $L(a) = \{u : (u, a) \in E(R)\}$. If not all $L(a)$ are in the same core in G , if some vertices in $L(a)$ are not in G , or if $|L(a)| = \emptyset$, a is removed from R .

3.3.5 The BuildPlus algorithm

The BUILDPLUS algorithm reduces the input P and D to the input \mathcal{G} of \mathcal{GC} . \mathcal{G} then be solved by GRAPHBUILD. By Lemma 3.2, the output of \mathcal{GC} is the output of \mathcal{TC} .



Chapter 4

Restricted constraint graph

We replace the constraint graph with a restricted constraint graph. A *restricted constraint graph* is a constraint graph, with the modification of intermediate vertices, red bundles, and aqua bundles. A restricted constraint graph G'_k is *associated* with a constraint graph G_k if G_0 and G'_0 are reduced from the same input P and D of \mathcal{TC} , and G_k, G'_k are derived from G_{k-1}, G'_{k-1} , respectively, with the same vertices set removed.

4.1 Intermediate vertices

Lemma 4.1. *Let P and D be the input of \mathcal{TC} , and $\mathcal{G}_0 = (G_0, \phi, Q_0, R_0)$ be the input of \mathcal{GC} reduced from P and D . Let $\mathcal{G}'_k = (G'_k, \phi, Q'_k, R'_k)$ be a restricted constraint graph associated with \mathcal{G}_k . Then (1) $L(G_k) = L(G'_k)$; (2) for each $x \in L(G_k)$, $\text{outdegree}_{G_k}(x) = 0$ if and only if $\text{outdegree}_{G'_k}(x) = 0$; (3) for each two $x, y \in L(G_k)$, x, y are connected in G_k if and only if x, y are connected in G'_k .*

Furthermore, $|V(G''_k)| + |E(G''_k)| = O(\|P\|)$, where $G''_k = G \setminus V_a(G)$ and the reduction of G''_k can be done in $O(\|P\|)$.

Proof. For each internal node $u \in T$, if $|L(u)| > 1$ and $|L(\text{child}(u))| > 1$, an intermediate vertex t and edges $\{(x, t) : x \in L(u)\} \cup \{(t, y) : y \in L(\text{child}(u))\}$ is built in G' for each $T \in P$. Let $\phi(t) = \text{indigo}$, a new color in ϕ .

After the removal of sources S , \mathcal{G}_{k-1} becomes \mathcal{G}_k , \mathcal{G}'_{k-1} becomes \mathcal{G}'_k . If after the removal of S , the indegree of t will become 0, t is added to S . Note that the vertices in S are still in the same core.

For each $y \in \{y \in L(G_{k-1}) : (t, y) \in E(G_{k-1})\}$, y has indegree 0 in G_k . For each two $y, z \in \{y, z \in L(G_{k-1}) : (t, y), (t, z) \in E(G_{k-1})\}$, y, z are connected in G_k , if and only if y, z are connected in G'_k . \square

4.2 Bundles

Lemma 4.2. For the restricted constraint graph $\mathcal{G} = (G, \phi, Q, R)$ reduced from P and D , the total update time for red edge removal step and of aqua label removal step can be done in $O(h \log^2 h)$ time and $O(h)$ space, where $h = \|P\| + \|D\| + |V(G')| + |E(G')|$, and G' be the subgraph of G induced by $L(G)$.

4.2.1 Red bundles

A red bundle tree T is a non-empty three-layered rooted tree in Q . If $v \in V(T)$ has no outdegree, v has exactly two ancestors.

Terms. Let $L(T)$ be the leaves of T , $\rho(T)$ be the root of T , $I(T)$ be the internal nodes of T , $P_T(x)$ be the parent of x in T . For a subset $L' \subseteq L(T)$, let $\Psi_T(L')$ be the red bundle tree induced by L' and all the ancestors of L' in T . For a vertex $v \in I(T)$, $v \notin V(G) \cup L(Q) \cup V(R)$. For a leaf $x \in L(T)$, $x \in L(Q)$. A red bundle tree T is *removed*

if $I(T)$ is removed from Q . For a set of blue vertices S , let $RT(S)$ be the red bundle trees T with $L(T) \cap S \neq \emptyset$, and for a graph G , let $RT(G)$ be $RT(L(G))$.

Lemma 4.3. *For a constraint graph $\mathcal{G} = (G, \phi, Q, R)$, An blue vertex x has no red edge if and only if x has no edges from red bundle trees.*

For a constraint graph $\mathcal{G}_k = (G_k, \phi, Q_k, R_k)$ and a blue vertex $x \in L(G_k)$, Let

$$\Lambda_k(x) = \{y : (x, y) \in E(Q_k)\}.$$

Lemma 4.4. *Let \mathcal{G}_0 be the input constraint graph, and $\mathcal{G}_k = (G_k, \phi, Q_k, R_k)$ be a constraint graph with $k > 0$ from \mathcal{G}_{k-1} by removal of a vertex set. If a data structure W keeps $\Upsilon_0(x) = \Lambda_0(x)$, and $\Upsilon_k(x) = \Lambda_k(x)$, and the vertex set $\{x : |\Upsilon_k(x)| = 0, |\Upsilon_{k-1}(x)| \neq 0\}$ can be known, for each blue vertex $x \in L(G_k)$, then W can replace the red edges.*

Proof. In BUILDPLUS, a vertex has no red edges if $|\Lambda_k(x)| = 0$. □

Property. When a core C becomes its children \mathcal{C} , the following property is maintained:

Property 4.5. *for each red bundle tree $T \in RT(C)$, T becomes the forest $child(T) = \{\Psi_T(L') : L' = L(T) \cap L(C'), C' \in \mathcal{C}\}$, where the $I(T)$ could be duplicated or moved in the induction while L' stays the same vertices set.*

The reduction In the reduction from \mathcal{TC} to \mathcal{GC} , for each node p with outdegree greater than 1, a red bundle tree T with root r is built. For each child v of p , a vertex k is created as a child of r . And for each label $x \in L(v)$, a leaf x is created as a child of k .

The removal of red bundle tree For a red bundle tree T , if $\rho(T)$ has only one child, remove T .

Let

$$\Upsilon_k(x) = \bigcup_{T \in RT(x)} \Upsilon_k(x, T).$$

We prove that $\Upsilon_k(x) = \Lambda_k(x)$, and the vertex set $S = \{x : |\Upsilon_k(x)| = 0, |\Upsilon_{k-1}(x)| \neq 0\}$

Proof. Let $\mathcal{G}_k = (G_k, Q_k, R_k)$ with $k \geq 0$ be a constraint graph, and after the removal of some vertices $S \in \mathcal{G}_k$ with $S \in V(C)$, $C \in G_k$, let the resulting constraint graph be \mathcal{G}_{k+1} .

Terms. Let $T \in RT(Q_k)$ be a red bundle tree and $x \in L(Q_k)$ be a blue vertex.

Let $\Upsilon_k(x, T)$ be

$$\begin{cases} \{y \in L(T) : P_T(x) \neq P_T(y)\}, & \text{if } x \in T, \\ \emptyset, & \text{if } x \notin T. \end{cases}$$

And $\Upsilon_k(x) = \bigcup_{T \in RT(Q_k)} \Upsilon_k(x, T)$ with $k \geq 0$.

Let $\Lambda_0(x) = \{y : (x, y) \in Q_0\}$, and $\Lambda_0(x, T) = \Upsilon_0(x) \cap L(T)$. Let $\Lambda_k(x) = \{y : (x, y) \in E(Q_k)\}$ and $\Lambda_k(x, T) = \Lambda_{k-1}(x, T) \setminus \{y : (x, y) \in Q_{k-1}, (x, y) \notin Q_k\}$ with $k > 0$.

Initialization. When the constraint graph \mathcal{G} is built, for each $x \in L(Q_0)$ and each $T \in RT(Q_0)$, $\Lambda_0(x) = \Upsilon_0(x)$ and $\Lambda_0(x, T) = \Upsilon_0(x, T)$.

For each $x \in L(Q_k)$ and each $T \in RT(Q_k)$, $\Lambda_k(x) = \bigcup_{T \in RT(Q_k)} \Lambda_k(x, T)$.

Induction step. When \mathcal{G}_k becomes \mathcal{G}_{k+1} , let $C \in C(G_k)$ be the core with vertex set removed, $\mathcal{C} = \text{child}(C)$. Let x be a blue vertex, $x \in L(C)$ before removal, and $x \in L(C')$ after removal for $C' \in \mathcal{C}$. And Let $T \in RT(Q_k)$ and $T \in RT(x)$, T becomes $T = \text{child}(T)$ in Q_{k+1} .

Let $T' = \Psi_T(L')$ with $L' = L(T) \cap L(C')$. Since T' is the subtree of T induced by

L' and its ancestors, $\Upsilon_{k+1}(x, T') = \Upsilon_k(x, T) \cap L(C')$. Since all the red edges (x, y) with $y \notin L(C')$ are removed, $\Lambda_{k+1}(x, T') = \Lambda_k(x, T) \cap L(C')$. With $\Upsilon_k(x, T) = \Lambda_k(x, T)$, we have $\Upsilon_{k+1}(x, T') = \Lambda_{k+1}(x, T')$.

An edge $(P_{T'}(x), x)$ in T' is removed if and only if $\rho(T') = \emptyset$. And $\rho(T') = \emptyset$ if and only if $\Upsilon_{k+1}(x, T') = \emptyset$. So, $RT(x) = \emptyset$ if and only if x has no red edges.

□

For a constraint graph $\mathcal{G} = (G, \phi, Q, R)$, let $\|RT(Q)\| = \sum_{T \in RT(Q)} |V(T)|$.

Lemma 4.6. *For the constraint graph build by the reduction, $\|RT(Q)\| = O(r)$. And the the reduction for the red bundles cost $O(r)$ time and space where $r = \|P\|$.*

Proof. For each internal node u in each tree $T \in P$, a bundle tree with $\cup_{v \in \text{child}(u)} |L(v)|$ leaves are created. So, $\|RT(Q)\| = O(r)$. And it is a linear time reduction, so the time and space for the reduction are both $O(r)$. □

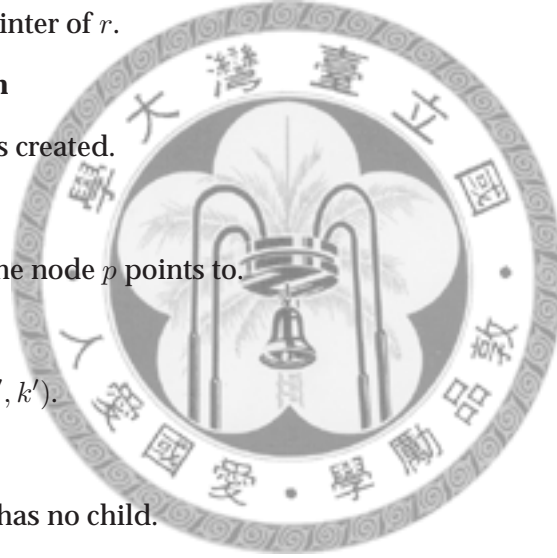
Lemma 4.7. *The red bundle trees has the partial update property. And if each blue vertex updates red bundle trees at most $O(t)$ times, the red bundles can be maintained in $O(rt)$ time with $O(r)$ space where $r = \|RT(Q)\|$ for the input constraint graph $\mathcal{G} = (G, \phi, Q, R)$.*

When a core C becomes its children \mathcal{C} , let each blue vertex $x \in C'$ updates each red bundle tree $T \in RT(x)$ by the following steps, where $C' \in \mathcal{C} \setminus \{C''\}$ and $C'' \in \mathcal{C}$ is an arbitrary core.

For a red bundle tree T , let each node $u \in I(T)$ has a pointer, which is null when created, and reset to null after each child core updated.

Algorithm 2 $\text{update}(x, T)$

- 1: $k \leftarrow P_T(x), r \leftarrow P_T(k)$.
 - 2: Let q be the pointer of k .
 - 3: **if** q is null **then**
 - 4: A node k' is created.
 - 5: **else**
 - 6: Let k' be the node q points to.
 - 7: **end if**
 - 8: Create edge (k', x) .
 - 9: Let p be the pointer of r .
 - 10: **if** p is null **then**
 - 11: A root r' is created.
 - 12: **else**
 - 13: Let r' be the node p points to.
 - 14: **end if**
 - 15: Create edge (r', k') .
 - 16: Remove (k, x) .
 - 17: Remove k if k has no child.
 - 18: Remove r if r has no child.
-



After all vertices in C' updated, reset all visited pointers to null.

We then prove Lemma 4.7.

First, the bundles are initially maintained, $T = \Psi_T(L)$, $L = L(T) \cap L(C_0)$, since all blue vertices are initially in the single core C_0 of G_0 .

Correctness After the removal of a set of vertices in C , each C' is updated.

For each C' and $L' = L(T) \cap L(C')$, we show that $\Psi_T(L')$ is created. For $w \in I(T) \setminus I(\Psi_T(L'))$, w will not be visited in the update of C' . For $u \in I(\Psi_T(L'))$ and the child v which visits u , in the first visit of u , the pointer of u is null, and a new node u' and the edge (u', v) are created as the duplication of u and (u, v) . And in the subsequent visit of u , the same u' is used and (u', v) is created. So, after all leaves in L' update, $\Psi_T(L')$ is created.

For the rest part of T , all leaves of $L(T) \cap L(C'')$ are removed from T since all $C' \in \mathcal{C} \setminus C''$ are updated. And all internal nodes with outdegree 0 is removed. Also, no new edges are created to the $V(\Psi_T(L''))$. So, the tree $\Psi_T(L'')$ is induced, where $L'' = L(T) \cap L(C'')$. So the red bundles are maintained.

So, the red bundles has the partial update property.

Complexity For each x , constant space for each T is kept, since the number of ancestors, the edges and the pointers of x and its ancestors is $O(1)$ though each update. And after C' is visited, the list of visited pointers can be released, the time and space of the list is constant for each pointer. Each update of x and T costs constant time. Since each x is updated at most t times, the total time for the update is $\sum_{x \in L(Q_0)} RT(x)$.

For each bundle tree with the outdegree of root 0, it is removed only once. So, the time for the removal is linear to the size of the red bundles in addition to the time above.

So, if each blue vertex updates red bundle trees at most $O(t)$ times, the red bundles can be maintained in $O(rt)$ time with $O(r)$ space.

4.2.2 Aqua bundles

An aqua bundle tree $T \in R$ is a non-empty two-layered rooted tree consists of a root $\rho(T)$ and two sets of leaves $L(T) \subseteq L(R)$ and $V_r(T) \subseteq V_r(R)$. For T , $|L(T)| > 0$ and $|V_r(T)| > 0$. For a set of blue vertices S , let $AT(S)$ be the red bundle tree T with $L(T) \cap S \neq \emptyset$, and for a graph G , let $AT(G)$ be $AT(L(G))$. And a function $x = \alpha(T)$ from $AT(R)$ to $N \cup \{0\}$.

Property. When a core C becomes its children \mathcal{C} , the following properties is maintained:

Property 4.8. For each $u \in L(T)$ is removed form the constraint graph, remove u from T , too. For each $T \in AT(C)$, if $|L(T) \cap L(C)| < |\alpha(T)|$, T is removed.

4.2.3 Replacing the aqua edges with aqua bundles

Without the loss of generality, a relative divergence date $div(L) < div(L')$ can be described as the compact form $div(L, x) < div(L', x')$ with $x = x' = 1$.

Let $indegree(a)$ be the indegree of the aqua vertex a in R .

Lemma 4.9. For a constraint graph $\mathcal{G} = (G, \phi, Q, R)$ and each relative divergence date d in the form $div(L, x) < div(L', x')$, let the created set of aqua vertices be A . And let T be the corresponding aqua bundle.

T can replace A and the edges created in the reduction form $div(L, x) < div(L', x')$.

Lemma 4.10. For a constraint graph $\mathcal{G} = (G, \phi, Q, R)$ and each relative divergence date d in the form $div(L, x) < div(L', x')$, let the created set of aqua vertices be A .

At least one $l \in L'$ is not connected to the aqua vertices $a \in A$ with all $a \in A$ with $indegree(a) = 0$ removed in the version of aqua vertices, if and only if all $l \in L'$ is not

connected.

Proof. For $div(L, x) < div(L', 1)$, it is true, since all aqua vertices connect to the same set of L' . For $div(L, 1) < div(L', 0)$, it is true, since all aqua vertices a with (a, y) and $y \in L'$ has same set of edges $\{(b, a) : b \in L\}$ in R .

For $div(L, 0) < div(L', 0)$, it is transformed into the set $Div = \{div(L, 0) < div(\{v\}, 1)\}$, for $v \in L'$. For each $d \in Div$, d has the property. Since all are not connected if at least one is is not connected. We only need to show for an v and each $u \in L$, $\{div(u) < div(v, 1)\}$ has the property. And it is $div(L, 0) < div(z)$, and shown in above case. That finishes the proof. □

We then prove Lemma 4.9.

Proof. We prove T can replace A and the edges, by showing if each $a \in A$ with $indegree(a) = 0$ removed, then a blue vertex $b \in L'$ is connected to no aqua vertices $a \in A$ in the version of aqua vertices, if and only if T is removed.

By Lemma 4.10, we only need to show that no blue vertex $b \in L'$ is connected to any aqua vertices $a \in A$ in the version of aqua vertices, if and only if T is removed.

For $div(L, x) < div(L', x')$, if all aqua vertex $a \in A$ created by each $d \in Div$ has $indegree(a) = 0$, then T can be removed, where Div is the expanded relative divergence dates $\{div(l) < div(l'')\}$.

If T can be removed, then all $a \in A$ created by each $d \in Div$ has $indegree(a) = 0$.

For $div(L, x) < div(L', 1)$, in both version, an aqua vertex or aqua vertices are created and connected to $l \in L'$ which keep l in the same core. And for $div(L, x) < div(L', 0)$, an aqua vertex or a set of vertices are created for each $l \in L'$. In the both

cases they keep the $A' \subseteq L'$ with $|A'| = \max(1, x'|L'|)$ in the same core and prevent all $l' \in A'$ becomes sources.

That finishes the proof. □

The reduction The creation of aqua vertices and the edges for relative divergence dates is replaced by the steps here.

Let $\text{div}(L, x) < \text{div}(L', x')$ be a relative divergence date. If $x' = 0$, an aqua vertex a_i is created for each $z_i \in L'$. If $x' = 1$, an aqua vertex a is created. Let the set of created aqua vertices be A . And an aqua bundle T is built with a root ρ and the leaves $L(T) = L$, $f(T) = \max(x|L|, 1)$, and $V_a(T) = A$ where A is the set of aqua vertices created for this relative divergence date.

We then prove Lemma 4.9.

Proof. For $x = 1$, T is removed if some blue vertices in L are removed, or two blue vertices in L are in different cores. It is the same condition of removing edges on the aqua vertices in the version of aqua edges. So, all blue vertices created by $\{\text{div}(L) < \text{div}(\{z\})\}$ with $z \in L'$ are free to be removed.

For $x = 0$, T is removed if and only if all blue vertices in L are removed. In the version of aqua edges, for $d \in \{\text{div}(y, 1) < \text{div}(L', x')\}$ with $y \in L'$, all sets of aqua edges for d have edge removed in R if and only if all L' have no edge from aqua vertex created from d . □

For a constraint graph $\mathcal{G} = (G, \phi, Q, R)$, let $\|AT(Q)\| = \sum_{T \in AT(Q)} |V(T)|$.

Lemma 4.11. *For the constraint graph build by the reduction, $\|AT(Q)\| = O(s)$. And the the reduction for the red bundles cost $O(s)$ time and space where $s = \|D\|$.*

Proof. For each relative divergence date $\text{div}(L, x) < \text{div}(L', x')$, an aqua bundle is built with $|L| + |L'|$ leaves. And the aqua vertices and edges created in G and R are at most $O(|L'|)$. So, $\|AT(Q)\| = O(s)$. It is a linear time reduction, so the time and space is $O(s)$. \square

Lemma 4.12. *The aqua bundle trees has the partial update property. And if each blue vertex updates aqua bundle trees at most $O(t)$ times, the aqua bundles can be maintained in $O(st)$ time with $O(s)$ space where $s = \|AT(Q)\|$ for the input constraint graph $\mathcal{G} = (G, \phi, Q, R)$.*

Proof. The removed blue vertex can be identified, and when the removed blue vertex a updates the each $T \in AT(a)$, the removal of $(a, \rho(T))$ can be done.

For each $C' \in \mathcal{C} \setminus C''$, and each $T \in AT(C')$, since each blue vertex $a \in L(C')$ updates, the total number of $L(T) \cap L(C')$ can be known after C' is updated, and T can be removed if $|L(T) \cap L(C')| < \alpha(T)$.

For the core C'' not updated, since each $T \in AT(C'')$ but $T \notin AT(C')$ are not affected, the total number of $L(T) \cap L(C'')$ will not be changed. \square

4.3 Proof

We now prove Lemma 4.2.

Proof. By Lemma 4.3 and Lemma 4.9, the red edge removal step and aqua label removal can be done by red bundles and aqua bundles.

With Lemma 4.6 and Lemma 4.11, if the update of a blue vertex is at most $O(t)$ time, then this can be done in $O(ht)$ time and $O(h)$ space.

And with Lemma 2.3, since red bundles and aqua bundles has partial update prop-

erty, the times of update for each blue vertex is $O(\log h)$ times, and if the framework in Lemma 2.3 can be done in $T(n, m)$ time and $S(n, m)$ space with graph $G = (V, E)$, where $n = |V|, m = |E|$, then the total update time for red edge removal step and of aqua label removal step can be done in $O(h \log h + T(h, h))$ time and $O(h + S(h, h))$ space, where $h = \|G\|$ with G the input constraint graph. \square



Chapter 5

Proof of Theorem 1.1

Proof. Let $\mathcal{G} = (G, \phi, Q, R)$ be a restricted constraint graph reduced from the input P and D of TREE COMPATIBILITY. Let G' be the subgraph of G induced by $L(G)$.

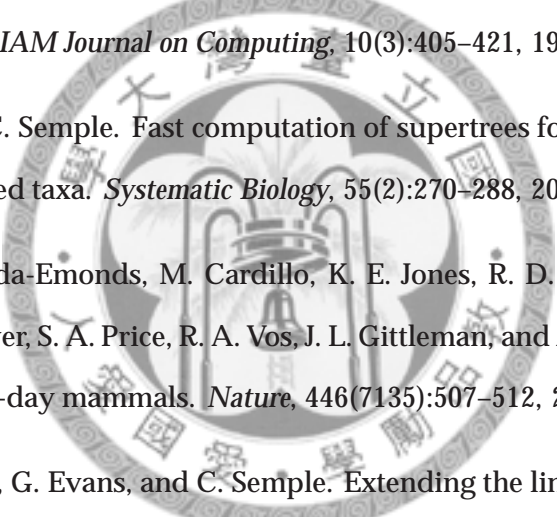
By Lemma 4.1, $|V(G')| + |E(G')| = O(\|P\|)$ and the reduction can be done in $O(\|P\|)$ time and space. Let $h = \|P\| + \|D\|$, we have $\|P\| + \|D\| + |V(G')| + |E(G')| = O(h)$. By Lemma 4.2, the red edge removal step and of aqua label removal step can be done in $O(h' \log^2 h')$ time and $O(h)$ space, where $h' = \|P\| + \|D\| + |V(G')| + |E(G')| = O(h)$.

And the time and space complexity of BUILDPLUS is $O(h + T(h))$ and $O(h + S(h))$, where h is the size of restricted constraint graph, $T(h)$ is the time for creating \mathcal{G}_k from \mathcal{G}_{k-1} with $\mathcal{G}_0 = \mathcal{G}$. $T(h) = O(h \log^2 h)$ and $S(h) = O(h)$.

So, the TREE COMPATIBILITY can be solved in $O(h \log^2 h)$ time and $O(h)$ space, where $h = \|P\| + \|D\|$.

□

Bibliography

- 
- [1] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.
- [2] V. Berry and C. Semple. Fast computation of supertrees for compatible phylogenies with nested taxa. *Systematic Biology*, 55(2):270–288, 2006.
- [3] O. R. P. Bininda-Emonds, M. Cardillo, K. E. Jones, R. D. E. MacPhee, R. M. D. Beck, R. Grenyer, S. A. Price, R. A. Vos, J. L. Gittleman, and A. Purvis. The delayed rise of present-day mammals. *Nature*, 446(7135):507–512, 2007.
- [4] M. Bordewich, G. Evans, and C. Semple. Extending the limits of supertree methods. *Annals of Combinatorics*, 10(1):31–51, 2006.
- [5] D. Bryant, C. Semple, and M. Steel. Supertree methods for ancestral divergence dates and other applications. In *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, pages 129–150. Kluwer Academic Publishers, 2004.
- [6] D. Bryant and M. Steel. Extension operations on sets of leaf-labelled trees. *Advances in Applied Mathematics*, 16(4):425–453, 1995.

- [7] M. Constantinescu and D. Sankoff. An efficient algorithm for supertrees. *Journal of Classification*, 12(1):101–112, 1995.
- [8] P. Daniel and C. Semple. Supertree algorithms for nested taxa. In *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, pages 151–172. Kluwer Academic Publishers, 2004.
- [9] P. Daniel and C. Semple. A class of general supertree methods for nested taxa. *SIAM Journal on Discrete Mathematics*, 19(2):463–480, 2005.
- [10] S. Even and Y. Shiloach. An on-line edge-deletion problem. *Journal of the ACM*, 28:1–4, 1981.
- [11] T. Griebel, M. Brinkmeyer, and S. Bocker. EPoS: a modular software framework for phylogenetic analysis. *Bioinformatics*, 24(20):2399–2400, 2008.
- [12] M. R. Henzinger, V. King, and T. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24(1):1–13, 1999.
- [13] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001.
- [14] M. P. Ng and N. C. Wormald. Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, 69(1–2):19–31, 1996.
- [15] R. D. M. Page. Modified mincut supertree. In *Proceedings of the 2nd International Workshop on Algorithms in Bioinformatics*, Lecture Notes in Computer Science 2452, pages 537–551, 2002.

- [16] C. Semple, P. Daniel, W. Hordijk, R. D. Page, and M. Steel. Supertree algorithms for ancestral divergence dates and nested taxa. *Bioinformatics*, 20(15):2355–2360, 2004.
- [17] C. Semple and M. Steel. A supertree method for rooted trees. *Discrete Applied Mathematics*, 105(1–3):147–158, 2000.
- [18] C. Semple and M. Steel. *Phylogenetics*. Oxford University Press, 2003.
- [19] M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9(1):91–116, 1992.
- [20] M. Steel, A. W. M. Dress, and S. Bocker. Simple but fundamental limitations on supertree and consensus tree methods. *Systematic Biology*, 49(2):3630–368, 2000.
- [21] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 343–350, 2000.

