2022 届本科生学士学位论文　　学校代码：10269


華東師範大學

East China Normal University

# 本科生毕业论文

## 完美带权二叉树的最优转换问题

## Optimal conversion problem of perfect weighted binary tree


姓　　名：田瀚杰

学　　号：10181511314

学　　院：数学科学学院

专　　业：数学与应用数学

指导教师：吕长虹

职　　称：教授


2022 年 5 月

## 华东师范大学学位论文诚信承诺

本毕业论文是本人在导师指导下独立完成的，内容真实、可靠。本人在撰写毕业论文过程中不存在请人代写、抄袭或者剽窃他人作品、伪造或者篡改数据以及其他学位论文作假行为。

本人清楚知道学位论文作假行为将会导致行为人受到不授予/撤销学位、开除学籍等处理（处分）决定。本人如果被查证在撰写本毕业论文过程中存在学位论文作假行为，愿意接受学校依法作出的处理（处分）决定。

承诺人签名：　　日期：　　年　　月　　日

## 华东师范大学学位论文使用授权说明

本论文的研究成果归华东师范大学所有，本论文的研究内容不得以其它单位的名义发表。本学位论文作者和指导教师完全了解华东师范大学有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅；本人授权华东师范大学可以将论文的全部或部分内容编入有关数据库进行检索、交流，可以采用影印、缩印或其他复制手段保存论文和汇编本学位论文。

保密的毕业论文（设计）在解密后应遵守此规定。

作者签名：　　导师签名：　　日期：　　年　　月　　日

# 目录

# 完美带权二叉树的最优转换问题

**摘要**：

在实际生活中，随着越来越多的问题都可以抽象成图进行处理，图挖掘成了我们理解并解决实际问题的重要手段。在本文中给我们将基站资源配置问题抽象为完美带权二叉树的最优转换问题，并提出了三类优化方法用于提升该问题的求解速度。

我们提出的三类方法分别是：多角度剪枝、优化搜索方法和局部动态规划。多角度剪枝除了将当前处理的二叉树损失与最优解进行比较剪枝外，还估计了每次合并或分裂操作的下界进行剪枝，并且在搜索开始时寻找多个较优可行解用于高效剪枝；在搜索方法的优化上我们首先证明了在对节点进行分裂操作时，采用贪婪选择同样能够找到最优解，此外采用了广度优先搜索代替深度优先搜索方法，并且在每次处理后对待搜索树序列进行重新排序，在考虑深度的条件下采用贪婪的思想，使最优解或较优解更容易被提前搜索；局部动态规划只对二叉树的最后三层进行处理，将倒数第二层的排序问题转换为只需要对有序序列插入或移除两个元素，从而借用已有较优解高效计算在对其他节点操作的条件下的新较优解。上述三种方法共同显著提升了解决完美二叉树最优转换问题的效率。

我们对提出的优化方法进行了数值模拟，虽然在深度 $Depth$ 较小的树上因为该算法需要计算并储存更多的辅助变量导致优化不明显，但在 $Depth > 20$ 时该算法比原始方法快接近 500 倍，并且优化后的方法能够更加高效的解决相同深度二叉树转换中复杂度相对较低的问题，因此该优化相比原始方法有较大提升，证明了我们提出的优化的合理性。

**关键词**：二叉树, 剪枝, 贪婪搜索, 动态规划

# Optimal conversion problem of perfect weighted binary tree

Abstract：

In our lives, as more and more problems can be abstracted into graphs, graph mining has become an important tool for us to understand and solve practical problems. In this paper, we abstract the optimal conversion problem of base station resources of signal providers into the optimal conversion problem of perfect weighted binary tree, and propose three categories of optimization methods to improve the solution speed of this problem.

We propose three categories of methods: multi-angle pruning, optimization search methods, and local dynamic programming. In addition to comparing the currently processed binary tree with the optimal solution for pruning, multi-angle pruning also estimates the lower bound of each merge or split operation for pruning, and finds multiple optimal feasible solutions at the beginning of the search. In the optimization of the search method, we first proved that greedy selection can also find the optimal solution when splitting nodes. In addition, the breadth-first search method is used instead of the depth-first search method, and reorder the sequence of the search tree after each processing. What's more, I also adopt the greedy idea so that the optimal solution or the better solution is easier to be searched. Local dynamic programming only processes the last three layers of the binary tree, and the sorting problem of the second to last layers is transformed to the problem which we only need to insert or remove two elements from the ordered sequence, so as to borrow the existing better solution to efficiently calculate the new better solution. The above three methods together significantly improve the efficiency of solving the optimal conversion problem of a perfect binary tree.

We have carried out numerical simulations on the proposed optimization method. Although the optimization is not obvious in the tree with a small depth because the algorithm needs to calculate and store more auxiliary variables, the algorithm is better than when $Depth > 20$. The algorithm after optimization is nearly 500 times faster than the origin method. What's more, the optimized method can more efficiently solve the relatively low-complexity problem in binary tree transformation of the same depth. Therefore, the optimization is greatly improved compared with the original method, which proves that our proposed optimization is reasonable.

Keywords：binary tree, pruning, greedy search, dynamic programming

## 一、绪论

随着图在不同场景的广泛应用，图挖掘（Graph Mining）领域快速发展，其中图的相似度（Graph Similarity）在较多领域有着广泛的应用。图的相似性的度量主要有以下三类方法[1]：图的编辑距离及同态（Edit Distance/Graph Isomorphism）、特征提取（Feature Extraction）和迭代方法（Iterative Methods）。在图的编辑与同态方法中，Sanfeliu 和 Fu[2]发挥了重要作用，他们首次将编辑距离引入图中，Messmer 和 Bunke[3][4]引入了子图的编辑距离用于在一族图中高效寻找原图的相似图，Bunke 和 Horst[5]研究了成本函数（Cost Function）对图匹配的影响，TAI[6]指出可以通过三类对节点操作（Change、Delete、Insert）实现了从多叉树到多叉树的修正（Tree-to-Tree Correction Problem），并可证明该转换是多项式时间的，该方法是图编辑距离[7]（Graph Edit Distance）的基础想法。

我们的问题从基站（Base Station）资源配置最优转换问题中抽象而出：在一个总资源一定的基站，我们将信号设置为不同的档位，每个档位都有不同数量的容器，每个容器中都有一定数量的用户，有时我们需要重新分配不同信号档位的容器数量，我们的目标是使转换过程中影响的用户数量最少。经过抽象可知这是一个完美带权二叉树的最优转换问题，我们需要通过对节点的合并和分裂实现不同档位的转换，并且希望转换的总权重最小。而因为对节点的合并以及分裂均包含于 TAI 指出的对节点的三类操作中，因此该问题与上文中对图相似的问题有一定相似度，但因为实际问题只要求了转换后不同信号档位的容器数量，因此我们的转换不是从一棵完美二叉树到另一棵完美二叉树，而是将一棵完美二叉树转换到满足叶子节点序列的一个完美二叉树的集合中的某个元素，并且使转换的权重最小。因为满足叶子节点序列的完美二叉树集合的元素数量随着树的深度的增加而指数级增长，虽然从树到树的转换是多项式可解的[6]，但因为集合内的元素数量较大，该最优转换的求解难度陡然提升，并且很有可能不是多项式可解的问题，而为了解决实际问题，我们希望最优转换的求解更加高效。

本文提出了三类方法用于高效解决完美二叉树的最优转换问题：第一个方法是从多角度进行剪枝，除了简单的对比当前转换结果与历史最优结果剪枝之外，我们通过在搜索之前给出较好的可行解用于高效剪枝，并且在每次对节点进行合并或分裂操作后估计该操作对应的转换权重的下界用于提前剪枝；第二个方法是采用优化后的广度优先搜索（Breadth-First Search），我们对每次得到的子树集合根据权重和深度进行排序，使优化后的广度优先算法更容易找到最优或较优解，从而能够大量减少对树的枚举计算；第三个方法是一种局部的动态规划，这里局部的含义是我们只针对树的最后三层进行处理，我们将存储部分之前对节点操作计算得到的结果并进行微调得到另一操作的最优结果，以减少枚举过程中带来的冗余计算。

虽然我们在该最优转换问题中进行了大量算法层面以及代码层面的优化，但该方法

仍然不能在多项式时间内解决这个问题，因为它本身很有可能是 NP-Hard 的。我们将在最后提出一些对该问题的思考以及后续工作的方向。

本文的章节安排如下：第二部分主要介绍这个问题的实际背景，第三部分介绍抽象后的具体数学问题是完美带权二叉树的最优转换，第四部分总结了相关问题的研究状况，并给出了该问题的一种基础求解方法，第五部分主要讨论我们给出的三类优化方法，第六部分给出优化后的数值模拟结果，第七部分对本文进行总结并提出一些问题以及对未来工作方向的思考。

## 二、问题背景

随着手机的逐渐普及，人们对高质量通信有着越来越高的要求，作为信息与通信的基础设施提供商，如何向用户提供高速且稳定的通信服务成了他们面临的难题之一，其中最重要的就是基站处信号资源的分配问题。

### （一） 基站信号资源配置背景

如图 2-1 所示，总资源一定的基站可以发射不同档位的无线电信号，各个档位的信号的容器需要的资源不同，而每个容器内有一定数量的用户。例如在一个总资源为 16 的基站中，第零档位信号的每个容器需要资源 16，第一档位信号需要资源 8，第二档位信号需要资源 4，以此类推，图 2-1 中给出了一种将基站总资源充分利用的分配方式，每个容器上的数字表示该容器所含的用户数量。

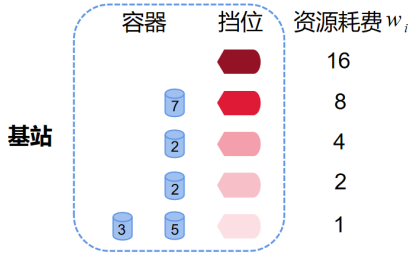


图 2-1　基站资源配置示意图

基站通常需要对现有的无线电信号进行重新编排，如图 2-2 所示，需要将档位 1 的一个容器，档位 2 的一个容器，档位 3 的一个容器和档位 4 的两个容器重新分配为档位 1 的一个容器和档位 2 的两个容器，我们希望该次调整影响到的用户总数最少，因此如何重新分配基站资源就成了首先需要考虑的问题。

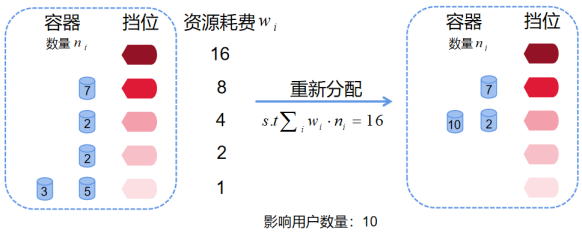


图 2-2　基站资源重新配置配置示意图

### （二） 基站资源配置的二叉树抽象

在开始之前，我们首先对完美带权二叉树进行定义：

**定义 2.1 (完美带权二叉树)**　完美带权二叉树是指叶子节点带有权重的完美二叉树（perfect binary tree），这里完美二叉树是指每个节点均含有 0 或 2 个孩子节点的二叉树。

在实际情况下，基站的总资源以及每个容器所占的资源都是 2 的幂次，且相邻档位容器的资源耗费为两倍关系，这引导我们考虑一个二叉树模型，因此我们将上述具体问题抽象为一个完美带权二叉树的最优转换问题。我们首先注意到基站的信号资源分配满足等式：

$$\sum_{i=0}^{d} w_i \cdot n_i = W$$

其中 $d$ 为信号档位总数，$w_i$ 为信号档位 $i$ 的容器所占的资源且满足 $w_i = 2^{d-i}$，$n_i$ 为信号档位 i 的容器数量，W 为基站总资源。

我们发现完美的二叉树的叶子节点同样满足上述等式，而且每个容器内的用户数量可以用完美的二叉树的叶子节点的权值表示，因此我们可以将该基站的当前状态抽象成一颗完美的带权二叉树，图 2-3 给出了将上文基站初始状态抽象为二叉树的示意图。

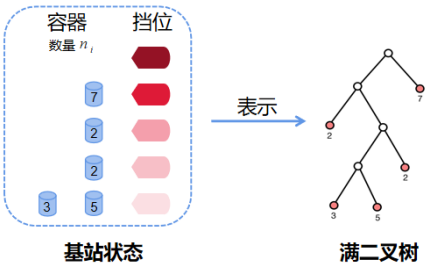


图 2-3　基站的完美带权二叉树表示

我们将基站不同档位容器的数目视为二叉树的叶子节点数量，那么基站状态的变化可以由完美二叉树通过对节点的合并与分裂实现叶子节点序列的变化表示，如图 2-4 所示，我们对树的节点进行合并表示基站从初始状态到最终状态的变化。其中，对叶子节点进行分裂操作以及对非叶子节点进行合并操作均会影响其对应的用户，但因为调整信号对用户的影响是一次性的，因此合并和分裂操作影响的用户数量均只计算一次。

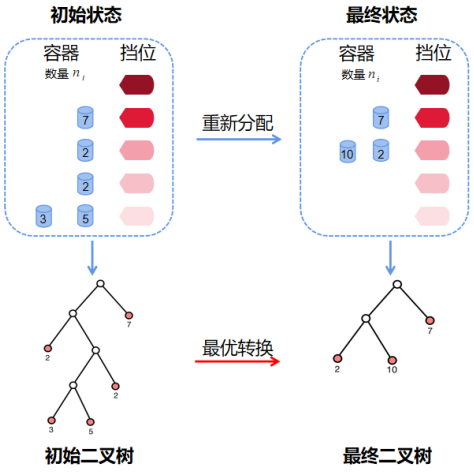


图 2-4　基站二叉树表示的最优转换

## 三、完美带权二叉树的最优转换

由上节可知，我们的问题是通过对完美二叉树进行节点的分裂和合并操作将其转换为满足叶子节点序列的新完美二叉树，并且使该转换的总损失最小。

我们注意到一个叶子节点序列 $l = [l_0, l_1, ..., l_d]$ 可以确定一个完美的二叉树的集合 $S_{tree}$，$\forall t \in S_{tree}$，均满足 $s$ 对应的叶子节点序列 $l_t = l$，因此我们不妨记集合满足的叶子节点序列为 $l_{S_{tree}}$。虽然初始状态所对应的树可以由实际情形确定，最终状态所对应的完美二叉树却不能被确定，因此由图 3-1 所示，该问题是求解当前二叉树到一个二叉树集合中某个元素的转换的总损失。

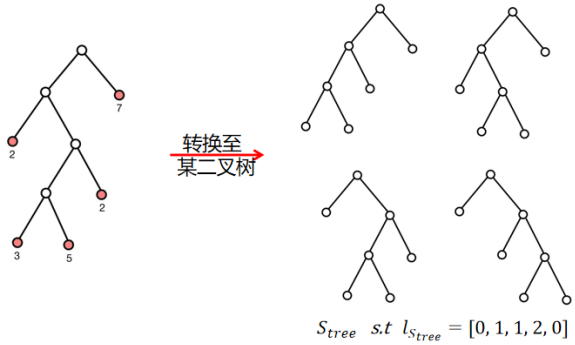


图 3-1　二叉树转换示意图

显然该转换涉及对节点的分类以及合并操作，我们首先对节点的分裂与合并进行如下定义：

**对叶子节点 $i$ 进行分裂（$S_i$）：**不妨设该节点位于二叉树的第 $k$ 层，该叶子节点的权重为 $w_i^l$，那么分裂操作对损失以及叶子节点序列的影响如下：

$$Cost_{new} = Cost_{old} + w_i^l$$

$$l_{new} = S_i(l_{old}) = [l_0, l_1, ...l_k - 1, l_{k+1} + 2, ..., l_d]$$

注意到因为权重仅计算一次，因此叶子节点 $i$ 的后继节点 $Successor(i)$ 中叶子节点的权重均为 0。

**对非叶子节点 $i$ 进行合并（$M_i$）：**同样设该节点位于二叉树的第 $k$ 层，其后继节点为 $Successor(i)$，树的叶子节点集合为 $L_{tree}$，不妨设该节点后继节在 $k$ 层之后每层的叶子节点数量为 $l_{k+1}^c, l_{k+2}^c, ...l_d^c$，那么合并操作对损失以及叶子节点序列的影响如下：

$$Cost_{new} = Cost_{old} + \sum_{j \in Successor(i) \cap L_{tree}} w_j^l$$

$$l_{new} = M_i(l_{old}) = [l_0, l_1, ...l_k + 1, l_{k+1} - l_{k+1}^c, l_{k+2} - l_{k+2}^c, ..., l_d - l_d^c]$$

因为非叶节点的权重可以提前计算，因此我们不妨记非叶子节点 $i$ 的权重为 $w_i^n = \sum_{j \in Successor(i) \cap L_{tree}} w_j^l$ 便于表示。

由上述节点分裂与合并得到的转换等式，我们可以将我们的问题表示如下：

$$min\ \ Cost = \sum_{i \in Split = \{i_1, i_2, \ldots i_k\}} w_i^l + \sum_{j \in Merge = \{j_1, j_2, \ldots j_t\}} w_j^n$$

$$s.t \quad S_{i_1} \circ S_{i_2} \circ \cdots S_{i_k} \circ M_{j_1} \circ M_{j_2} \cdots M_{j_t}(l^{tree}) = l^{obj}$$

因为合并节点对后续层的节点有较大的影响，该问题变得异常难以解决，甚至可能是 $NP - Hard$ 问题从而无法在多项式时间内找到最优解，在下两节中我们首先会对相关问题的研究状况进行总结，接着会对该问题的实际求解提出三类优化方法，使该问题的求解时间得到大幅优化。

## 四、相关问题的研究状况

在树的最优转换问题上前人已经进行了研究并得到了一些成果：Sanfeliu and Fu[2]首先提出在图上使用编辑距离，该方法通过统计需要重新打标签、加入、删除的节点和边来计算从一张图到另一张图的距离。在本文中我们考虑的是一颗完美带权二叉树的转换问题，TAI[6]指出，我们可以在多项式时间内计算出初始二叉树到另一二叉树的转换权重，但随着树的深度的增长，满足叶子节点序列的树集合元素个数 $\#(S_{tree})$ 会指数级增长，这对存储是个巨大的困难。纵然优一些启发式算法可以解决该问题，但并不能保证找到的解是全局最优。因此我们在实际计算中不考虑枚举所有满足目标的完美二叉树，而是直接考虑对初始二叉树进行节点的分裂以及合并使之满足叶子节点序列。

而考虑对节点的分裂与合并操作使二叉树满足叶子节点序列的做法，我们已有如下深度优先搜索算法，如 $Algorithm1$ 所示：

**Algorithm 1** DFS-Tree-Transformation

**Data:** Current Tree $T$, Current Iteration $i$, Max Depth $d$, Current Cost $C_{current}$, Current Leaves list $l$, Target Leaves list $l^{obj}$
**Result:** Optimal Cost $Cost$
**if** $i = d$ **then**
  $Cost \leftarrow C_{current}$
  return $Cost$
**end**
// Calculate the leaves and nodes index in depth $i$
$Leaves \leftarrow$ **LeavesCal(**$i$**,** $T$**)**
$Nodes \leftarrow$ **NodesCal(**$i$**,** $T$**)**
**if** $l_i = l_i^{obj}$ **then**
  $C_{current}$ = **DFS-Tree-Transformation(**$T$**,** $i+1$**,** $d$**,** $l$**,** $l_{obj}$**)**
**else if** $l_i < l_i^{obj}$ **then**
  // Choose the index of leaves to be split
  **for** *index in* ***Choose(****Leaves***,*** $l_i^{obj} - l_i$***)*** **do**
    $T.split(index)$
    $C_{current}^{index}$ = **DFS-Tree-Transformation(**$T$**,** $i+1$**,** $d$**,** $l$**,** $l_{obj}$**)**
  **end**
  $C_{current} = min_{index}\{C_{current}^{index} + \sum_{i \in index} w_i^l\}$
**else**
  // Choose the index of nodes to be merged
  **for** *index in* ***Choose(****Nodes****,*** $l_i - l_i^{obj}$***)*** **do**
    $T.merge(index)$
    $C_{current}^{index}$ = **DFS-Tree-Transformation(**$T$**,** $i+1$**,** $d$**,** $l$**,** $l_{obj}$**)**
  **end**
  $C_{current} = min_{index}\{C_{current}^{index} + \sum_{i \in index} w_i^n\}$
**end**
$i \leftarrow i + 1$
return $C_{current}$

但该方法本质上和枚举等同，计算了每一种从当前二叉树到目标二叉树的转换

方式并计算权重，再在所有权重中取最小。这种方法的时间复杂度会随着树的深度的提升指数增加，并且因为算法过程种出现了组合数 $Choose(Leaves, l_i^{obj} - l_i)$ 以及 $Choose(Nodes, l_i - l_i^{obj})$，导致最坏情况下很有可能无法计算出结果，因此我们需要对上述算法进行优化。

## 五、优化算法

在本节中，我们提出三种优化方法：从多个角度进行剪枝、采用优化后的广度搜索算法、局部动态规划。

### （一） 多角度剪枝

#### 1. 对比可行解进行剪枝

假设我们在转换问题上已经有了一个可行解 $C_{feasible}$，那么当我们在枚举分裂或合并节点时，如果当前损失已经大于上述可行解，即：

$$C_{current} + \sum_{i \in index} w_i \geq C_{feasible}$$

则停止搜索。

#### 2. 估计每次节点操作的下界提前剪枝

对第 k 层选取部分节点 $\{I_k\}$ 合并或分裂得到当前 $C_{current}$ 后，估计该操作的下界（考虑下一层的操作）：

**分裂操作（需要分裂 $t$ 个叶子节点）**：选取 $k+1$ 层权重最小的 $t$ 个叶子节点的权重 $w_i^l, i = 1, 2..t$，则 $k$ 层合并 $I_k$ 的操作下界为 $C_{current} + \sum_{i=1}^{t} w_i^l$;

**合并操作（需要合并 $t'$ 个非叶子节点）**：选取 $k+1$ 层权重最小的 $t'$ 个非叶子节点的权重 $w_i^n, i = 1, 2...t'$，则 k 层合并 $I_k$ 的操作下界为 $C_{current} + \sum_{i=1}^{t'} w_i^n$。

从而在搜索时，我们可以提前估计对节点操作的下界，与当前可行解 $C_{feasible}$ 对比进行剪枝。

#### 3. 搜索开始前寻到较优可行解

容易注意到，在每次分裂或合并节点时如果均采用贪婪搜索的方法可以快速给出一个可行解，但遗憾的是在某些极端情况下该方法找到的解可能很差，图 5-1 给出了这样的一个例子。

在该例中，如果采用贪婪方法，那么最终损失 $C = 196$，但是最优解 $Cost = 100$。为了解决该问题，我们提出了另一种提前计算可行解的方法，在该方法中，我们希望固定权重较大的子树使之不被分裂或合并，从而避免上述极端情形。

不妨假设需要对 k 层节点进行合并（共 K 层）：现有非叶子节点 $NL_k = \{i_1 : w_{i_1}^n, i_2 : w_{i_2}^n, i_3 : w_{i_3}^n ...\}$ 为一哈希表并按非叶子节点的权值降序排列，且存下每个 $i_t$ 所对应的子树的叶子节点序列 $nl_{i_t} = [l_{k+1}^{i_t}, l_{k+2}^{i_t}, ...l_K^{i_t}]$，$k+1 \sim K$ 的目标叶子节点序列为 $[l_{k+1}^{obj}, l_{k+2}^{obj}, ...l_K^{obj}]$，为了保证权重较大的子树得以保留，我们不断向集合 $S_{unchange}$ 中加入 $NL_k$ 中权重最大的子树位置直到不能再加入，过程中保证 $S_{unchange}$ 中的子树的

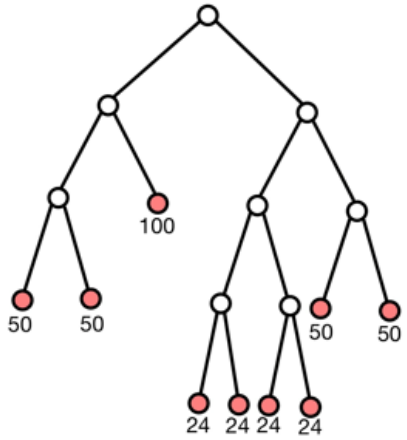


图 5-1　贪婪搜索反例

$k+1 \sim K$ 中每层的叶子节点数之和均小于该层的目标叶子节点数，例如当 $S_{unchange} = \{I_1, I_3, I_5\}, \sum_{i_t \in S_{unchange}} l_p^{i_t} \leq l_p^{obj}, p = k+1, k+2, ...K$，但若再加入 $i_7$（$i_7$ 为权重第四大的子树位置）得到 $S_{unchange} = i_1, i_3, i_5, i_7$ 后，$\exists p\ , s.t \sum_{i_t \in S_{unchange}} l_p^{i_t} \geq l_p^{obj}$，则算法停止。我们保持 $S_{unchange}$ 中的子树不动，解决一个原问题形式相似的 *Subproblem*：

在子问题中，当前非叶子节点序列为 $NL_k.keys() \backslash S_{unchange}$，目标叶子节点序列为 $[l_{k+1}^{obj} - \sum_{i_t \in S_{unchange}} l_{k+1}^{i_t}, l_{k+2}^{obj} - \sum_{i_t \in S_{unchange}} l_{k+2}^{i_t}, ..., l_K^{obj} - \sum_{i_t \in S_{unchange}} l_K^{i_t}]$

该方法可以一定程度上解决贪婪方法无效剪枝的问题：在子问题中，它考虑了权重较大的子树的结构后，一定程度上保证了某些权重中等的子树也被纳入了合并的考虑空间，而不是像贪婪方法中直接选取最小的几个子树进行合并而不管剩余子树的结构。因此该方法在某些情形下能比贪婪方法找到一个更好的可行解用于后续剪枝。

在实际求解子问题时，我们常采取贪婪选择。

## （二） 优化搜索方法

### 1. 对分裂节点采用贪婪选择

我们可以证明，对分裂操作采用贪婪选择依然可以得到最优解，在此之前我们先证明如下引理：

**引理 5.1**　假设两棵二叉树需要转换到同一叶子节点序列 $l^{obj}$，且已经满足 $l_i^1 = l_i^2, i = 1, 2, ..., k-1$，且当前损失 $C_{current}^1 = C_{current}^2$，在第 $k$ 层这两棵树有相同的叶子节点以及非叶子节点对应的子树，那么一定有 $Cost^1 = Cost^2$，即二叉树的最优转换损失与同一层中节点的顺序无关。

**证明**　反证法，假设在第 $k$ 层有两种不同的节点排序，使得最优损失 $Cost^1 < Cost^2$，因为第 $k$ 层仅仅是节点顺序不同，我们可以将对第一棵树第 $k$ 层节点的操作同样作用于

第二棵树（分裂操作用值对应，合并操作用子树对应），那么可以知道增加的损失相同，且第 $k+1$ 层的节点除了顺序也相同，由归纳得知 $\forall p, k \leq p \leq d$，第 $p$ 层的节点除了顺序均相同，增加的损失也相同，又 $C^1_{current} = C^2_{current}$，因此 $Cost^1 = Cost^2$，与第二棵树最优损失大于第一棵矛盾。

证毕。 □

从而我们有如下定理：

**定理 5.1** 假设二叉树第 $k$ 层需要进行节点分裂操作，且已经满足 $l_i = l_i^{obj}, i = 1, 2, ..., k-1$，那么对第 $k$ 层贪婪选取要分裂的节点同样可以得到最优解。

**证明** 假设第 $k$ 层叶子节点为 $L_k = \{i_1 : w^l_{i_1}, i_2 : w^l_{i_2}, ..., i_t : w^l_{i_3}\}$，我们需要从中选出 $s$ 个进行分裂操作，因此第 $k+1$ 层非叶子节点对应子树不改变，增加 $2s$ 个权重为 0 的叶子节点，因此无论分裂哪 $s$ 个叶子节点，$k+1$ 层均有相同的叶子节点和非叶子节点对应的子树，由引理 5.1 可知 $k+1$ 层之后的最优损失与顺序无关，因此为了总损失最小，我们应该在第 $k$ 层选取权重最小的 $s$ 个叶子节点进行分裂操作。综上我们在分裂节点时采用贪婪选择仍然可以得到最优解。

证毕。 □

因此，我们在搜索时对于分裂操作均采用贪婪选择从而极大减少计算。

2. **广度优先搜索**

因为在深度优先搜索时会出现组合数，但其中大部分都是无意义的探索，从而会浪费大量的计算，因此我们不考虑深度优先搜索，而采用广度优先搜索并用排序提前找到最优解。

我们首先建立一个待考虑子树集合 $S_{subtree}$，初始化结合中只有根节点，每次迭代中我们都会从中挑选出 $S_{subtree}$ 中 $C_{current}$ 最小的子树，不妨设该子树的前 $k$ 层已经满足叶子节点序列要求，即 $l_i = l_i^{obj}, i = 1, 2, ..., k$，我们考虑生成下一层节点。我们枚举出所有分裂（合并）生成的子树并更新该子树的 $C_{current}$，将这些子树加入到子树集合 $S_{subtree}$ 中进入下一次迭代。

该方法能更快找到最优解或较优解，从而配合剪枝等其他方法能够极大降低复杂度，具体算法见最后 $Algorithm2$。

### （三） 局部动态规划

对于该需要深度或广度搜索的问题，我们很自然的想法就是是否可以借助动态规划的思想对计算过程进行简化，但很遗憾的是因为该问题的子问题不一致，因此很难完全采用动态规划方法。但鉴于动态规划的想法为利用现有子问题结果高效解决新问题，我

们这里借用动态规划的想法进行优化，这里称为局部动态规划是因为我们只对树的倒数三层进行类似动态规划的处理。

因为树的倒数第二层无论合并还是分割都应该采取贪婪策略，因此算法在最后三层的耗费主要在多次排序上，我们希望减少这样的排序成本。不妨假设之前所有层已经合并或分割到目标叶子节点，考虑最后三层节点，因为叶子节点和非叶子节点是 1-1 对应的，因此我们下文考虑非叶子节点。

假设当前二叉树最后三层非叶子节点数量为 $N_{-3}, N_{-2}, N_{-1}$，目标非叶子节点数量分别为 $N'_{-3}, N'_{-2}, N'_{-1}$，且 $N'_{-3} < N_{-3}$，(否则直接贪婪分割，这里考虑合并情况)，那么倒数第三层以一共有 $C_{N_{-3}}^{N_{-3}-N'_{-3}}$ 种合并方式，不妨将合并方式的集合记为 $S_{merge}, s \in S_{merge}$ 表示合并的非叶子节点的位置，$s = i_1, i_2, i_3 ... i_{N_{-3}-N'_{-3}}$。

$\forall s \in S,\ \exists s' \in S,\ s.t.\ len(s \cap s') = len(s) - 1$，即 $s$ 与 $s'$ 中的元素只有一个不相同，例如 $s = 1,2,3,5,7$ 则可取 $s' = 1,2,3,5,9$，并且 $S_{merge}$ 一定存在一种排序方式，使得 $\forall s$，一定可以找到 $s'$ 排在 $s$ 的前面。

**下面我们考虑利用合并 $s'$ 简化合并 $s$ 的计算:**

先考虑合并 $s'$，合并后对倒数第二层的叶子节点和非叶子节点的权重均进行排序：$L_{-2} = \{i_1 : w_1^l, i_3 : w_3^l, i_7 : w_7^l ...\}$, $NL_{-2} = i_2 : w_2^n, i_4 : w_4^n, i_5 : w_5^n ...$ 为两个哈希表，均按权重升序排列，则 $s'$ 层的分割只要选 $L_{-2}$ 中的前 $t$ 个（合并只要选 $NL_{-2}$ 中的前 $t'$ 个即可）。而合并 $s$ 与 $s'$ 的差别在于仅需要分别删除和添加一个深度为 3 的树，删除一颗树相当于从 $L_{-2}, NL_{-2}$ 中一共 pop 两个元素，添加一颗新树相当于向有序哈希表 $L_{-2}, NL_{-2}$ 中共 insert 两个新元素，在 $n$ 较大时上述操作快于排序，从而达到近似动态规划的效果。

综上，我们对该问题进行了如上的一些优化，最终的算法如 *Algorithm*2 所示，其中上述的优化方法均在注释中标出，该优化的数值结果参见下一节。

**Algorithm 2** BFS-Tree-Transformation(Optimization)

**Data:** Current Tree $T$, Target Leaves list $l_{obj}$
**Result:** Optimal Cost $Cost$

```
// Calculate the suboptimal Cost
Cost ← Suboptimal(T)
S_tree ← {root}
while S_tree ≠ ∅ do
    tree_min ← Findmin(S_tree)
    S_tree.pop(tree_min)
    if tree_min.depth = d − 3 then
        // Use local dynamic method
        Cost_new ← LocalDynamic(tree_min)
        Cost ← min{Cost, Cost_new}
    else
        k ← tree_min.depth
        news ← {}
        Leaves, Nonleaves ← NodeCal(tree_min)
        if l_k^obj < #(Leaves) then
            // Use greedy method to find the leaves to be split
            index ← Greedychoose(Leaves, #(Leaves) − l_k^obj)
            tree_min.split(index)
            tree_min.cost = tree_min.cost + Σ_{i∈index} w_i^l
            // pruning
            if not pruning(tree_min) then
                news.add(tree_min)
            end
        else
            // Choose the index of nodes to be merged
            for index in Choose(Nonleaves, l_k^obj − #(Leaves)) do
                tree_min^copy ← deepcopy(tree_min)
                tree_min^copy.merge(idnex)
                tree_min^copy.cost = tree_min^copy + Σ_{i∈index} w_i^n  // pruning
                if not pruning(tree_min^copy) then
                    news.add(tree_min^copy)
                end
            end
        end
        S_tree.extend(news)
    end
end
return Cost
```

## 六、数值结果

在本节中我们给出上述优化方法的数值结果并对结果进行分析。

因为该实际问题的现有解决方案是采取深度优先搜索并在搜索过程中进行简单剪枝，分裂叶子节点时采用贪婪方法，因为我们将该方法作为标的与我们提出的优化方法进行比较。

我们对深度 $Depth$ 从 4 至 24 的完美二叉树的最优转换问题进行数值模拟，每轮我们随机生成 100 棵完美二叉树，叶子节点的权重为 $[0, 200]$ 内的随机整数，并随机生成目标叶子节点序列求解该最优转换问题。我们分别统计这 100 次问题求解的平均时长 $Time_{mean}$、时长标准差 $Time_{std}$、最坏时长 $Time_{max}$ 和最优时长 $Time_{min}$，因为两种算法的平均时长可能不在一个量级，因此我们采用 $\frac{Time_{std}}{Time_{mean}}$ 衡量算法识别困难与简单问题的能力，该值越大说明处理较为困难和较为简单问题之间的时间差距越大，因此可以从侧面反映出算法识别问题难易的能力。

平均计算时长 $Time_{mean}$ 与指标 $\frac{Time_{std}}{Time_{mean}}$ 随深度变化图像分别如图 6-1 与图 6-2 所示，可以看出在深度 $Depth$ 在 8 之前现有解决方法有更短的计算时长。这是可以预料到的，因为我们新设计的算法在计算过程中比原方法需要计算并储存更多的辅助变量，因此在深度较浅的二叉树最优转换问题中速度会稍慢。但是随着二叉树深度的增加，原有算法的计算时间几乎呈指数级增加，很快就到了我们不能承受的地步，而我们的算法显然增长较为缓慢，在深度为 20 以上的的二叉树转换问题中平均计算时间仅仅需要原方法的约 $\frac{1}{500}$，这显然是一个巨大的提升。而考虑指标 $\frac{Time_{std}}{Time_{mean}}$ 可以发现，我们优化后的算法在该指标上的表现随深度的增加变化不大，均在 0.75 ~ 1.50 附近浮动，原始方法的指标随着深度的增加快速降低到 0.2 以下，并且持续降低，说明我们的算法确实能够在问题比较简单的时候更高效的解决问题，相比之下原始方法对于不同难度的问题表现差异不大。

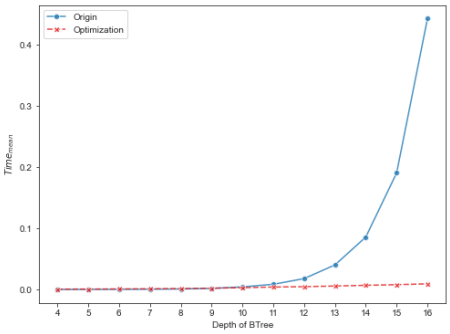


图 6-1　算法平均时长 $Time_{mean}$

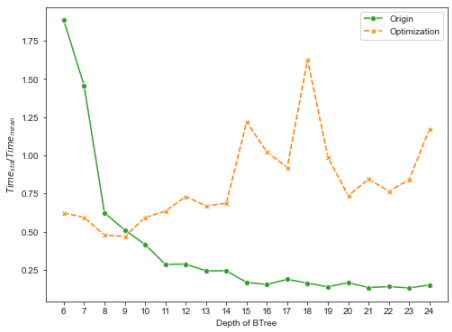


图 6-2　算法 $\frac{Time_{std}}{Time_{mean}}$ 指标

我们还比较了两种算法在不同深度最有转换问题中的最大和最小时长，反映了算法在面临最好以及最坏的情形需要花费的时间。如图 6-3 和图 6-4 所示，我们发现原始算

法的最坏以及最优时长均随着深度指数级增长，趋势和其平均时长较为类似。而我们优化后的算法比指数级缓慢得多，同样体现了本文算法的优势。

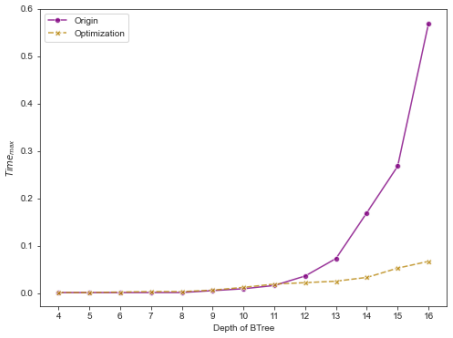


图 6-3　算法最坏时长 $Time_{max}$

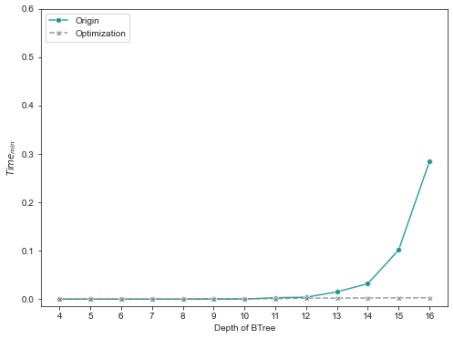


图 6-4　算法最优时长 $Time_{min}$

综上，无论是从平均时长还是从算法处理简单以及困难情形的能力来看，本文中的算法相比原始算法都有巨大的提升，在某些原始深度优先算法跑不出结果的情况，该方法仍然能够给出解，因此该方法有较强的实际应用价值。

## 七、结论

本文中我们针对具体问题提出了多角度剪枝、优化搜索方式以及局部动态规划的优化方法，并在数值模拟中证明了这些优化方法的有效性。其中，多角度剪枝除了将当前处理的二叉树损失与最优解进行比较剪枝外，我们还估计了每次合并或分裂操作的下界进行剪枝，并且在搜索开始时寻找多个较优可行解用于高效剪枝；在搜索方式的优化上我们首先证明了在对节点进行分裂操作时，可以直接采用贪婪选择同样能够找到最优解，此外采用了广度优先搜索代替更容易理解和实现的深度优先搜索方法，并且在每次处理后对待搜索树序列进行重新排序，在考虑深度的条件下采用贪婪的思想，使最优解或较优解能够在更靠前的时间点上被搜索到；为了高效利用已有较优解的局部结构，我们采用动态规划的方法对二叉树的最后三层进行处理，将倒数第二层的排序问题转换为只需要对有序序列插入或移除两个元素即可。上述三种方法共同显著提升了解决完美二叉树最优转换问题的效率。

在深度为 24 及以上的完美二叉树最优转换问题中，优化后的算法平均比现有解决方案快了约 500 倍，并且 $\frac{Time_{std}}{Time_{mean}}$ 指标较为稳定，而原始方法的该指标逐渐降低，说明明优化后的方法能够更加高效的解决相同深度二叉树转换中复杂度相对较低的问题，换句话说，该优化后的算法具有较强的问题辨别能力。

但遗憾的是，我们仍然没有给出该问题的多项式算法，随着树的深度的逐渐提升，即使速度远远快于原有算法，我们的优化方法的计算时间仍然会以接近指数的速度增长，这源于该问题本身可能是个 $NP-Hard$ 问题，我们之后的工作重点将会放在对该算法的进一步优化。

# 参考文献

[1] KOUTRA D, PARIKH A, RAMDAS A, et al. Algorithms for graph similarity and subgraph matching[J]., 2011, 17.

[2] SANFELIU A, FU K S. A distance measure between attributed relational graphs for pattern recognition[J]. IEEE transactions on systems, man, and cybernetics, 1983(3): 353-362.

[3] MESSMER B T, BUNKE H. Efficient error-tolerant subgraph isomorphism detection[C]//Shape, Structure and Pattern Recognition. [S.l. : s.n.], 1994: 231-240.

[4] MESSMER B T, BUNKE H. A new algorithm for error-tolerant subgraph isomorphism detection[J]. IEEE transactions on pattern analysis and machine intelligence, 1998, 20(5): 493-504.

[5] BUNKE H. Error correcting graph matching: On the influence of the underlying cost function[J]. IEEE transactions on pattern analysis and machine intelligence, 1999, 21(9): 917-922.

[6] TAI K C. The tree-to-tree correction problem[J]. Journal of the ACM (JACM), 1979, 26(3): 422-433.

[7] GAO X, XIAO B, TAO D, et al. A survey of graph edit distance[J]. Pattern Analysis and applications, 2010, 13(1): 113-129.

# 致谢

此时的我正坐在宿舍门口的草坪上，身旁各种球类运动的声音中夹杂着话语声，我将致谢推到了最后的最后才勉强下笔，是因为这篇类似对四年的总结让我有一种“当自己写完时，大学就真的结束了”的恍惚感，但随着交稿日期的临近，我不得不下笔完成整篇论文的最后一部分。想起刚看到的推送提到上海明天最高将达 33°C，夏天是不是真的来了呢？图书馆对面的毕业花有没有开呢？在最后两个月的时间里我还有机会走遍校园吗？明天会更好吗？当我开始思考“毕业”时，这些问题接踵而至，那么我索性从“入学”开始，回忆和思考一些已经确确实实发生过的事吧。

大一时我带着些许不甘进入了这所学校，倒不是对自己的高考成绩抑或是自招志愿有过多的抱怨，更多是被亲戚朋友以及同学诸如“为啥选择这所学校?”、“师范学校哇，你要去当老师吗?”、“学数学会不会出来找不到工作呀？”等种种问题轰炸产生的自我怀疑，虽然有解释过“我学的是非师范”、“学数学就业方向挺广的”等等，但从他们勉强的笑容中还是看出了这样的解释并没有太多说服力。但随着学期的开始这种情绪很快就消失了，我意识到身边有很多优秀的同学，老师也打破了我对“数学老师”的一些传统观念。陆俊老师和王丽萍老师是我本科阶段对数学认知的启蒙者，陆老师的课很有意思，时常一本正经的抛梗令人捧腹，他通过线上解答了我很多学业上的问题，让我在高代的学习过程中减少了许多困难；王老师有些许严厉，我仍记得有一回在课上玩游戏她直接走到了我的身旁盯着我，属实让我回忆起了被班主任统治的恐惧))。她也非常严谨认真，力求教会同学的过程中经常对问题进行推广，这极大的影响了我在学习数学时的思考方式。更重要的是两位老师从不吝啬对学生的理解和夸赞，这让我从原本的无限自我怀疑中获得了些许自信。

度过了被小学期、数科院赞助的新加坡游学（旅游）以及军训填充的满满当当的大一暑假，时间来到了大二。席卷而来的新冠疫情使人手足无措，武汉封城、高考延期等等有生之年事件仿佛在时刻提醒我 2020 终将是魔幻的一年，留学的计划也因此搁置，但也是在这一年里我对未来可能的方向做了诸多尝试。许忠好老师的概率论以及刘玉坤老师的数理统计将我领入统计的大门，相比纯碎的基础数学，她似乎更具有与世界相连的“烟火气”。梁金荣老师的数分三、复变函数，苗俊杰老师的实变函数是我联系数学与其他学科的重要桥梁。梁老师异常温柔仿佛永远不会生气，每次考试都会在办公室耐心评点我们的试卷，点评完必然还会加上一句鼓励；苗老师的风格让我记忆犹新，严格认真还酷爱和同学互动，被 que 到后连麦近半小时属实是让我再也不敢吱声))，但这不妨碍实变函数成为了我学习过的最重要的课程之一，是我学业和工作上的重要工具。双创方面感谢姚燕珊老师的指导，组员们在我安排不当时给予建议和理解，在我提出问题时能积极出谋划策，每当群里看到“收到!”的回复确实让人安心了很多。暑期实习中，小明老师帮助我对量化有了初步的了解，他的一句“不要沉迷技术，要思考更贴近现实的假设”提醒我在日常学习中时常抬起头想一想与业界的联系，实习结束后他仍会与我交流业界情况并给我的项目和实习提供一些思路，非常幸运能在第一份实习中便遇到那么 nice 的 mentor。

如果一定要说的话，大三应该是我的本科阶段最不平凡的一年了，在这一年中经历了太多的大起大落。从上学期考试失利，一个人坐在长椅上看月亮思考路在何方的万念俱灰，到下学期在图书馆偶然一瞥看到 Congratulations 的邮件，几乎快要从椅子上跳起来的激动万分，每一件发生的事、当时的场景以及怀有的情感仿佛历历在目，提醒着我这一切实实在在的发生过。上学期石芸老师的行为金融学和 BAB 因子相关课题让我学会利用另类视角去理解量化交易，鄙人愚钝，常常对一些涉及

经济学的定理理解不够深刻，石老师总会对我的提问进行详细的解释并指出理解的不当之处，令我受益匪浅。很快春天就来了，一二教旁的樱花树下站着三三两两，每个人的嘴角似乎都挂着笑容，我却因为在求职与申请之间反复横跳而有些疲惫。在申请的过程中感谢许忠好老师、刘玉坤老师、陆俊老师、梁金荣老师、石芸老师以及顾青老师帮我撰写和修改推荐信，因申请手续繁多且各个院校标准不一，我时常需要多次麻烦他们，他们从未表现出不耐烦并多次对我进行鼓励，我对此非常感激。在求职的过程中非常感谢张伦学长的帮助，他的分享让我对互联网不同职位有了最初的认识，他还帮助我关注面试流程、提醒我面试注意事项，让枯燥的四月因一句 Welcome to join Alibaba 而增添了些许颜色。

转眼间就来到了大学的最后一年，杭州的夏天与上海一样热，一开始畅想着下班之后或许可以去西湖边散散步，奈何公司的免费宵夜和打车不给我这个机会。因为诸多原因，当我开始实习时距离答辩仅只有一个月的时间，在我异常焦虑的情况下非常感谢 mentor 发散对我的指导和帮助，让我顺利在答辩之前完成了项目。此外还非常感谢答辩前帮我 mock 的摇竹、钲月、鲁猫、驹隙等老师，他们对我的项目以及展现方式提出了诸多宝贵意见，让我在面对面试官的提问时更加从容。在秋天快要结束的时候，我回到上海开始了另一段量化实习，这段实习极大刷新了我的认知并重建了我的方法论，让我深刻意识到当时的我有多么浅薄和渺小。在老板 Claude 的指导下，我确确实实加深了对统计、数学、计算机、心理等学科在二级市场上的应用的理解，也让我明白了一个厉害的 PM 应该具有什么样的能力。最后还要感谢我的毕设导师吕长虹老师，吕老师的离散优化选讲激发了我对该方向的兴趣，并在论文选题以及推进中给了我重要的指导建议，帮助我如期完成毕业设计。

更需要感谢的便是我的父母，当和同学聊起回家的话题时才想起除了疫情那年，我每年待在家中的时间两个巴掌似乎都数得过来，他们不止一次暗戳戳表现出对我的想念，却因为对我“可能”在忙的理解不会说出口，也从不会对我的回家时间或忘记接电话表现出不高兴，总是愿意在我低落的时候充当我的宣泄对象并不讲理的安慰我。要是今年暑期有空的话，就多回家几天吧。

在我心情跌入谷底的至暗六月，很感谢一位同学接受我的情绪垃圾并不断鼓励我。我不断回忆自己的面试，将自己想吐槽觉得丢人的点一而再再而三重复到自己都想吐的时候，这位同学总是会想方设法得告诉我“没关系的，真的一切都是最好的安排”，让我的负面情绪没有停留太久，不再后悔或不甘于已经发生过的事儿。

接着就不得不提我的学校华东师大，这里有宽敞明亮的图书馆，有价格实惠的学生宿舍，还有偶尔拉垮但大部分时候都很好吃的饭菜。在这里我还遇到了很有趣的人、很仗义的人、很尊敬的人和很喜欢的人，我常常因为一些些自闭疏于交际，很难将对他们的感激、感谢或是其他情感直白得说出口，但毋庸置疑的是他们确实成为了我记忆中不可分割的一部分，一辈子都不会忘记。

还有给我上过课的老师、帮助过我或对我进行知识输出的同事、一起完成作业或项目的同学、愿意给我指导的学长学姐……此时太多太多的人闪现过我的脑海，我真切得想把每个人每件事都记录下来，但限于篇幅只能就此作罢。

即使下午超过了三十度，晚风依然有些凉，夜跑的同学从身旁经过，草坪上坐着些女生似乎在开茶话会，疫情下的生活节奏被严重的打乱，但大家似乎不约而同适应了下来，似乎真的如徐燕婷老师所说：“学会处变不惊是我们一生的功课”。在这个不平凡的 2022 年的上海，在看到了一些令人难过的事情后，我意识到了活着可能便是一种极大的幸运，我对此非常感激。

明天似乎要降温了，校园里的那些花应该会在降温后再次开放吧。 Q.E.D.