**深圳大学研究生实验报告**

题目 Partition Problem 成绩

专业 计算机科学与技术 课程名称、代码 并行算法

年级 2020级 姓名 章涵艺

学 号 2060271030 时间 2020 年 1 月 15 日

任课教师 廖好

# Introduction

## Question

在数论和计算机科学中，分区问题或数分区是确定是否可以将给定的正整数多集S划分为两个子集S 1和S2的任务，以便使S1中的数字之和等于S2中数字的总和。尽管分区问题是NP完全的，但是仍然存在伪多项式时间动态规划解决方案，并且在许多情况下都有启发式方法可以最优或近似地解决该问题。因此，它被称为“最简单的难题”。 有一个分区问题的优化版本，它是将多集S划分为两个子集S 1，S 2，以使S1中元素之和与S 2中元素之和之间的差异最小。优化版本是NP难的，但在实践中可以有效解决。分区问题是两个相关问题的特例：

1. 在子集和问题中，目标是找到S的一个子集，其总和是输入给定的一定数W（分区问题是W为一半的特例S的总和）。
2. 在多路数字分区中，有一个整数参数k，目的是决定是否可以将S划分为相等和的k个子集（分区问题是k = 2的特殊情况）。
3. 但是，它与3分区问题完全不同：在该问题中，子集的数量不是预先固定的-应该是| S | / 3，其中每个子集必须恰好具有3个元素。 3分区比分区难得多-除非P = NP，否则它没有伪多项式时间算法。

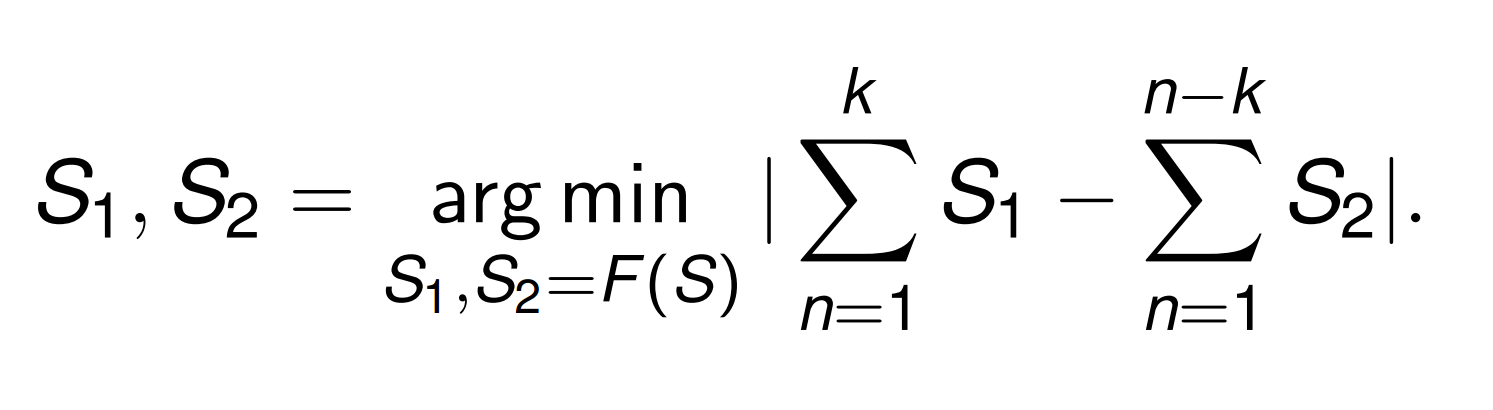
**1.2 Examples**

这里有三个简单的示例：

1. 给定S = {3，1，1，2，2，1}，该Partition Problem的有效解决方案是两组S1 = {1，1，1，2}和S2 = {2，3} 。
2. 给定S = {8，7，6，5，4}，，该Partition Problem的有效解决方案是两组S1 = {8，7}和S2 = {6，5，4} 。
3. 给定S = {13，12，11，10，9，8，7}，，该Partition Problem的有效解决方案是两组S1 = {13，12，10}和S2 = {11， 9、8、7} 。

# Problem Formulation

分区问题：给定一组整数S = {s1，s2，...，sn}，我们想找到一个函数F将S划分为两个子集S1，S2，使得S1之和之间的差为 S2被最小化。



# Algorithms

## Algorithm Classification

组合优化问题的算法可分为以下四类：

**（1）精确算法。**

精确算法是指能够求出问题最优解的算法。当问题的规模较小时，精确算法能够在可 接受的时间内得到最优解；当问题的规模较大时，精确算法一方面可以提供问题的可行 解，另一方面可以为启发式算法提供初始解，以便搜索到更好的解。

常用的精确算法有分支定界法、割平面法、动态规划法等。

**（2）近似算法。**

近似算法是指用近似方法来解决优化问题的算法，通常与 NP-hard 问题相关，由于无 法有效地在多项式时间内精确地求得最优解，所以考虑在多项式时间内求得一个有质量保 证的近似解。

贪婪算法、局部搜索算法、松弛算法、动态规划法等都可用于构建近似算法求解。

**（3）启发式算法。**

启发式算法是一种基于直观或经验构造的算法，能在可接受的计算成本内尽可能地逼 近最优解，得到一个相对优解，但无法预计所得解与最优解的近似程度。

启发式算法可分为传统启发式算法和元启发式算法，传统启发式算法包括构造性方 法、局部搜索算法、松弛方法、解空间缩减算法等。

**（4）元启发式算法。**

元启发式算法主要指一类通用型的启发式算法，它对启发式算法进行了改进，是随机 算法与局部搜索算法相结合的产物。这类算法的优化机理不过分依赖与算法的组织结构信息，可以广泛地应用到函数的组合优化和函数计算中。

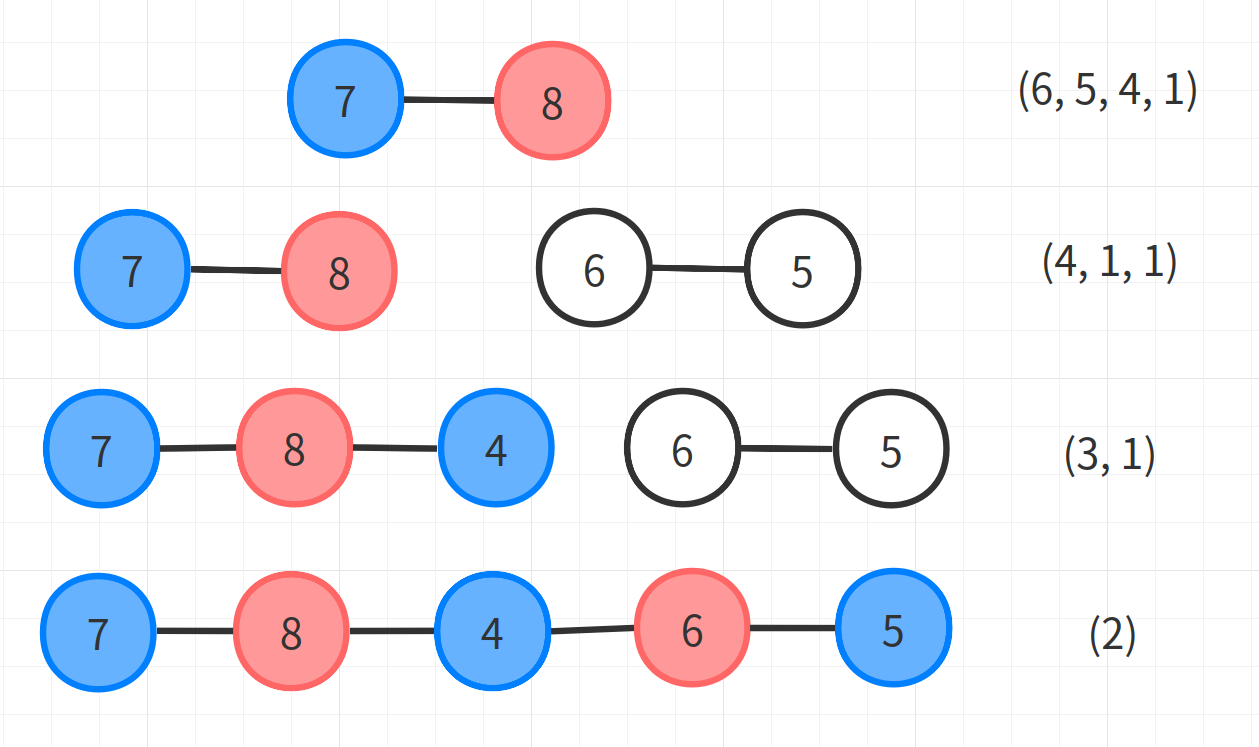
## 3.2 Approximation Algorithm

1.1中介绍，分区问题是多路分区和子集和的特殊情况。因此，可以通过针对每个问题开发的算法来解决。为多路分区划分开发的算法包括：

**（1）贪婪划分：**循环编号，并将每个数字放入当前总和最小的集合中。如果未对数字进行排序，则运行时间为O（n），并且近似比最大为3/2（“近似比”表示算法输出中的较大和，除以最佳分区中的较大和）。对数字排序将运行时间增加到O（n log n），并将近似比提高到7/6。如果数字在[0,1]中均匀分布，则近似比率最多是确定的，并且是期望值。

**（2）最大差分法**（也称为Karmarkar-Karp算法）按降序对数字进行排序，并用其差值重复替换数字，该算法是ckk算法的基础，其具体实例如下图所示。运行时复杂度为O（n log n）。在最坏的情况下，其近似比率相似-最高为7/6。但是，在一般情况下，它的性能要比贪心算法好得多：当数字以[0,1]均匀分布时，其近似率最多是期望值。它还在模拟实验中表现更好。 Multifit算法将二进制搜索与二进制装箱算法相结合。在最坏的情况下，其逼近率为8/7。

**KK [2] (Kurmarkur-Kurp heuristic) in (8, 7, 6, 5, 4)**



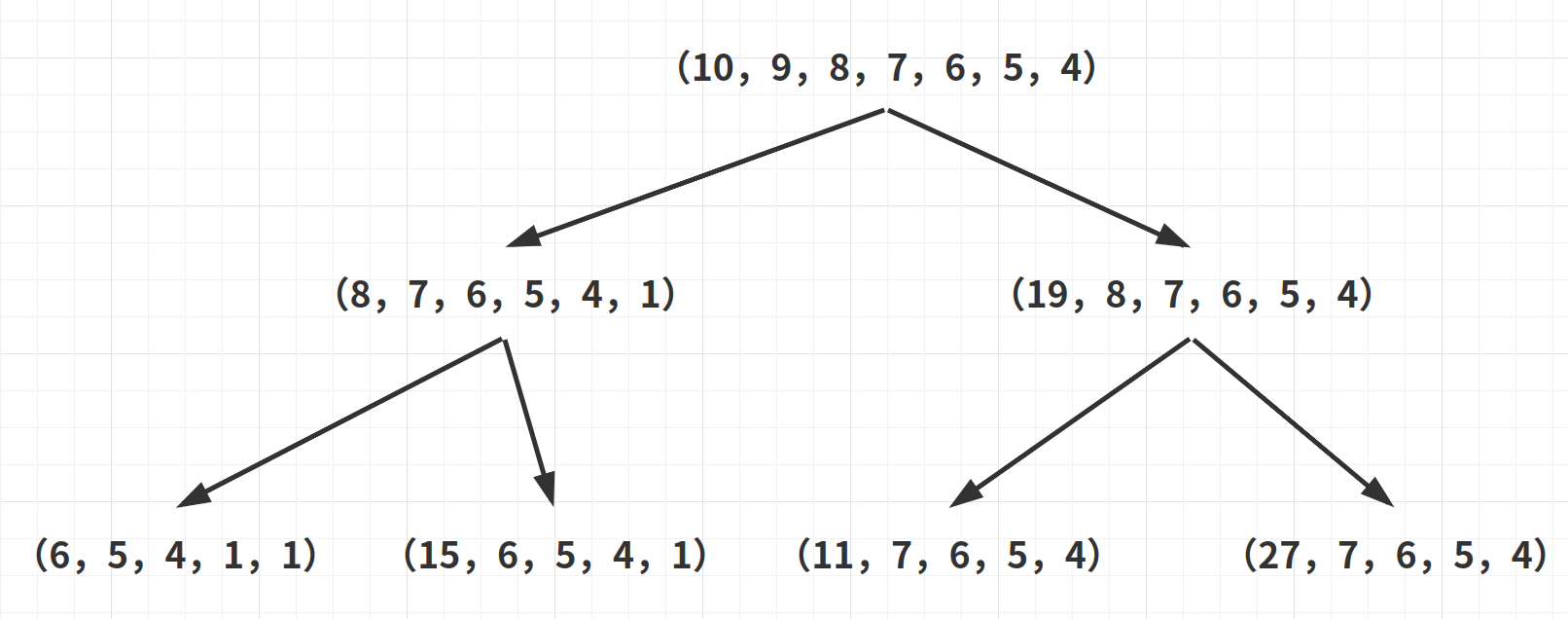
## 3.3 Exact Algorithm

## 同时该问题也存在精确的算法，总能找到最佳分区。由于问题是NP难题，因此此类算法通常可能花费指数时间，但在某些情况下可能会实用。为多向数字分区开发的算法包括：伪多项式时间数字分区占用内存，其中m是输入中的最大数字。

## **完全贪婪算法（CGA）**：通过构造二叉树来考虑所有分区。树中的每个级别对应于一个输入数字，根目录对应于最大数字，下面的级别对应于第二大数字，依此类推。每个分支对应于一个不同的集合，可以在其中放置当前数字。以深度优先的顺序遍历树仅需要空间，但可能需要时间。可以通过使用贪婪的启发式方法来改进运行时：在每个级别中，首先开发将当前数字放入总和最小的分支中。该算法首先找到通过贪婪数划分找到的解决方案，然后继续寻找更好的解决方案。这种想法的一些变体是针对子集和问题的全多项式时间逼近方案，因此也适用于分区问题。

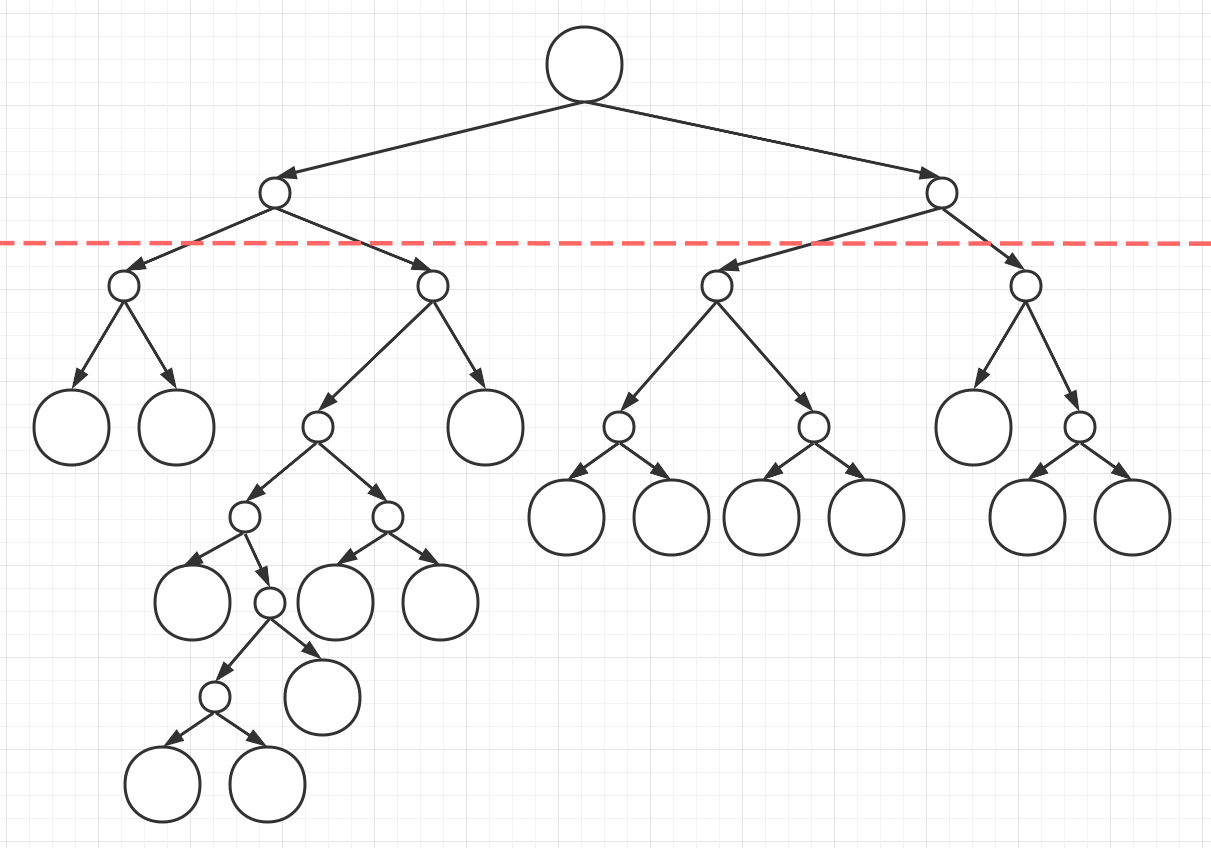
## **Complete Karmarkar-Karp算法（CKK）**：通过构造二叉树来考虑所有分区。每个级别对应于一对数字。左分支对应于将它们放在不同的子集中（即，用它们的差异替换它们），而右分支对应于将它们放在相同的子集中（即，用它们的和替换它们）。该算法首先找到通过最大微分法找到的解，但是随后近似算法继续寻找更好的解，**如下图所示**。在随机实例上，它的运行速度明显快于CGA。当存在相等的分区时，它的优势要大得多，并且可以达到几个数量级。实际上，如果数字最多具有12个有效数字，则CKK可以解决任意大小的问题。 CKK还可以作为任何时间的算法运行：它首先找到KK解，然后在时间允许的情况下逐渐找到更好的解（在最坏的情况下，可能需要指数时间才能达到最优）。它需要空间，但在最坏的情况下可能需要时间。

**CKK [1](Kurmarkur-Kurp heuristic) in (10, 9, 8, 7, 6, 5, 4)**



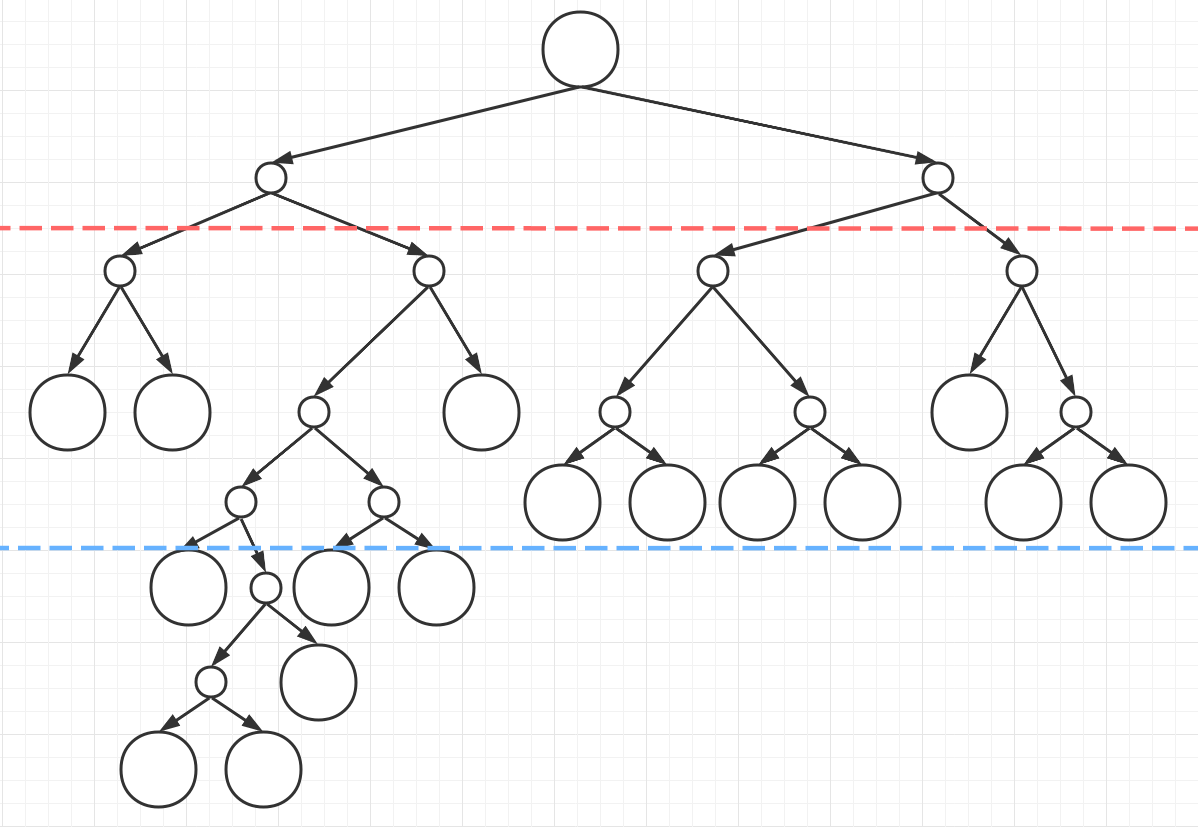
**3.4 Extend CKK with Parallel**

**对CKK算法进行并行化处理，**由于ckk算法是一个递归问题，在生成递归树的过程中对子问题开并行处理是相对直观的过程，并且在并行的过程中进行记录，最后得到最小值。并行过程如下图所示。



**3.4 Extend CKK with Parallel and CUT**

**利用**并行处理的特性，在满足误差范围的条件下，在CKK并行过程中对子问题进行剪枝，设定一个参数如 **0.7**，在70% 的并行都结束运行时，程序选择停止，即还剩下的30%分支停止运行，如图所示。



# Experiment

实验环境为C++；操作系统：Ubuntu 18.04；系统内存：32 G；

处理器： **64 Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz（16核）**

实验数据：伪随机数取值方式如下：

**#define ACONST 25214903917 /\* constant multiplier \*/**

**#define MASK 281474976710655 /\* 2^{48}-1 \*/**

**#define C 11 /\* additive constant \*/**

**seed\_init = 13070**

**for num:**

**seed = (ACONST \* seed + C) & MASK;**

根据同一取初始化循环产生200个数时：

最大值为281291068533640

最小值为385424385004

CPU是16核，一组实验开了32个线程

**每组实验运行10次，求得差值的和作为评判准确性的标准**

**时间：**

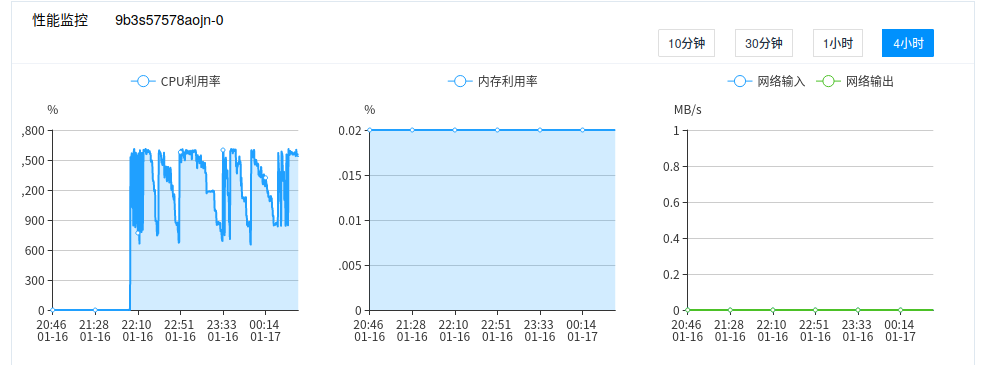
**每组实验的CPU消耗时间,【计算并行CPU总消耗时间（使用clock()）】**

每组实验的真实（实际）时间，开并行后真实时间会小于串行时间

**并行CPU总消耗时间 - （对应）串行消耗时间 = 开并行所消耗额外时间**

**（并行算法中，希望开并行所消耗额外时间很小，没有免费的午餐）**

**【当采用cut时，则不一定并行CPU总时间大于串行总CPU时间】**



**并行求解时，CPU利用率并不稳定**

## Five Important Experiment

**实验规模： 75个随机数为一组：110个为一组：每组间差5个，横纵向对比**

1. 串行运行 ckk 算法

输出文件到 norm.txt

1. 并行运行 ckk --- 开32个线程，对比时间

输出文件到 para.txt

1. 并行运行 ckk --- 开32个线程，cut 80%，对比时间，精度

输出文件到 para\_cut\_20.txt

1. 并行运行 ckk --- 开32个线程，cut 70%，对比时间，精度

输出文件到 para\_cut\_30.txt

1. 并行运行 ckk --- 开32个线程，cut 60%，对比时间，精度

输出文件到 para\_cut\_40.txt

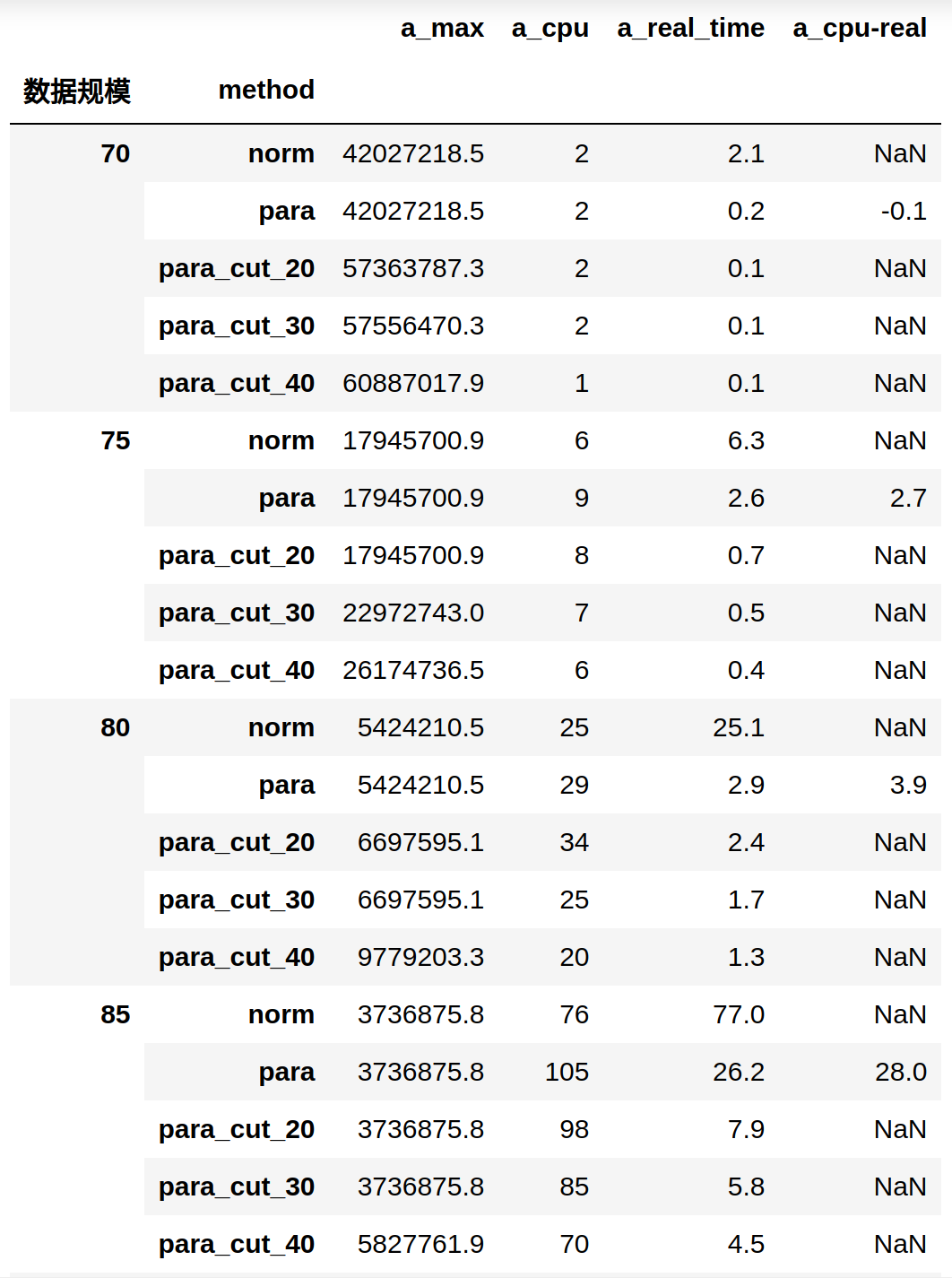
## Experiment Result Analysis

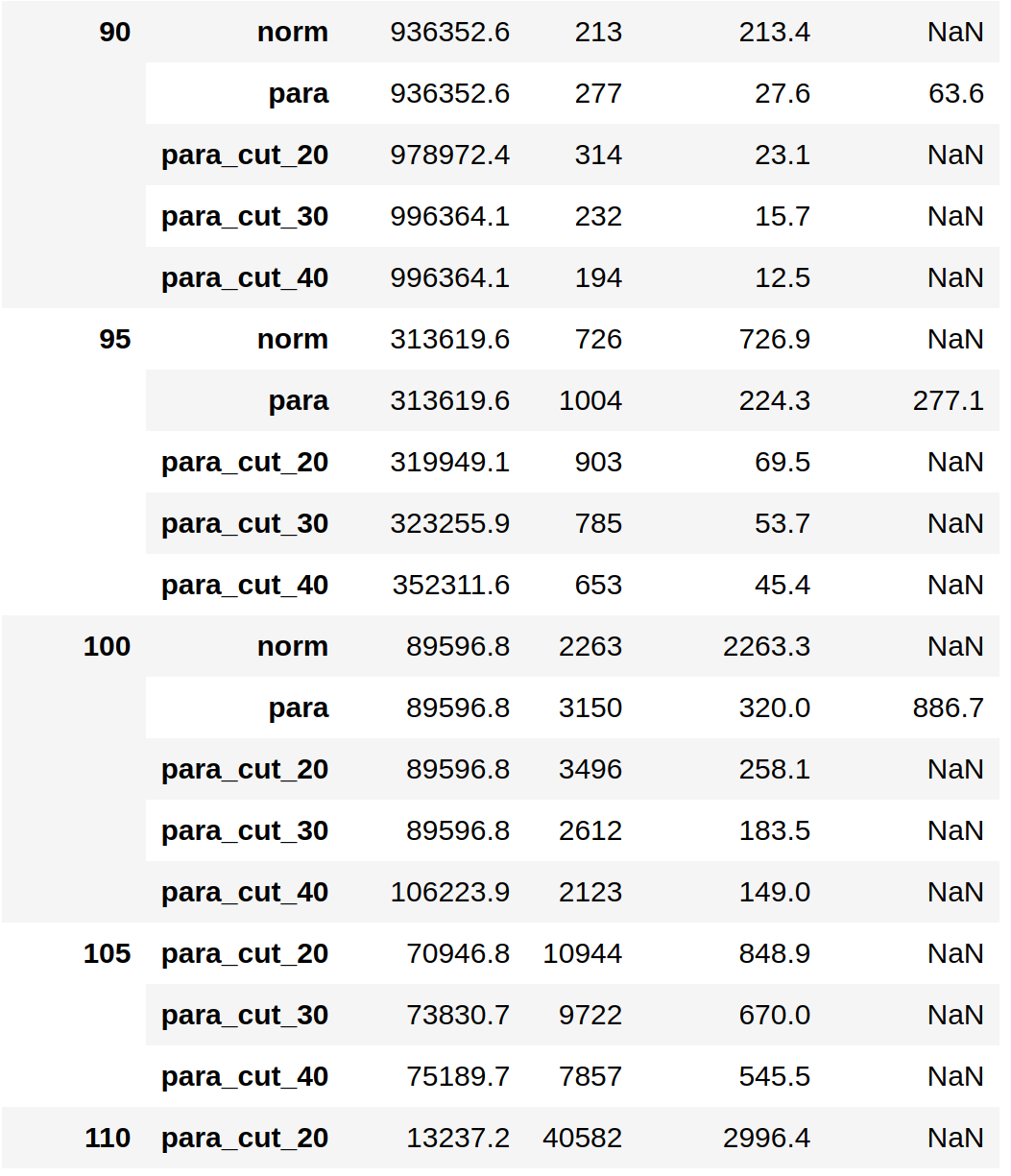
a\_cpu是求解十次实验，每一次实验总cpu消耗的平均时间

a\_real\_time 是求解十次实验，实验消耗的实际平均时间

a\_max 是求解十次实验，totalmax / 实验数，即平均difference，要求越小越好，用来判断算法精度

a\_cpu-real是求解十次实验，相同规模下，并行算法CPU消耗时间 - 串行CPU消耗时间，表示开并行所需要的时间

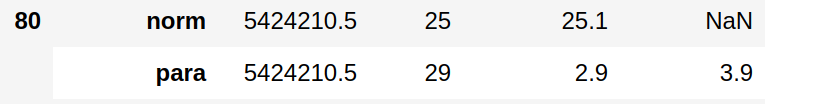




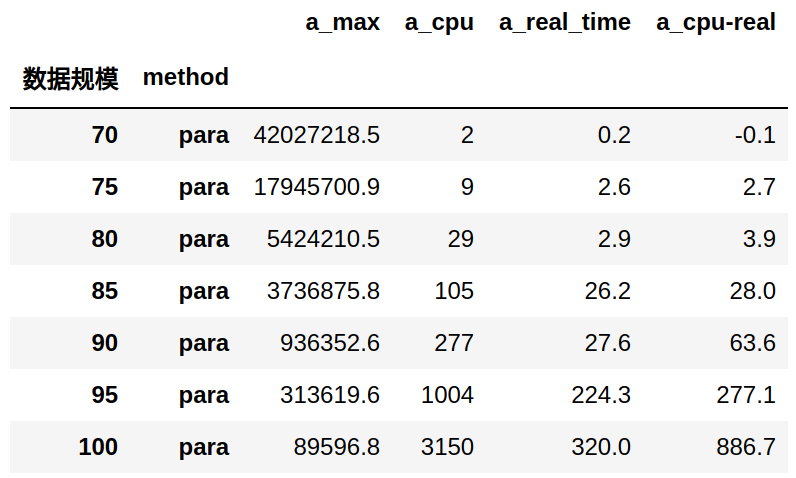
实验结论：

（1）ckk观察第三列或是第四列时间相关的变量，在method一定时，发现其时间复杂度随着规模变化，成指数增长。

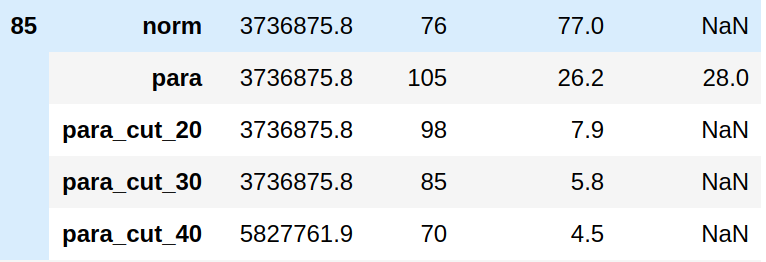
（2）并行化对算法速度的提升还是很明显的，但是某些时候并不稳定，可能是因为有些线程会很难终止，而这一点就是提出cut的原因，总体来说速度提升还是很明显，如图：

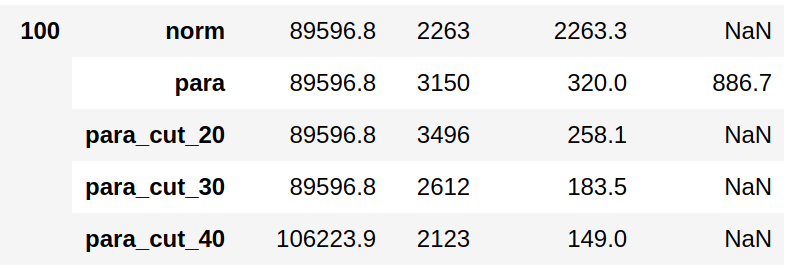


1. 开并行所需要的时间，也在随着问题规模的增大而增大，当然，这个时间平摊到各个cpu上所需要的时间又会减少很多了。如下，886.7s就是该规模该method下所有并行要比串行额外花费的总的时间。

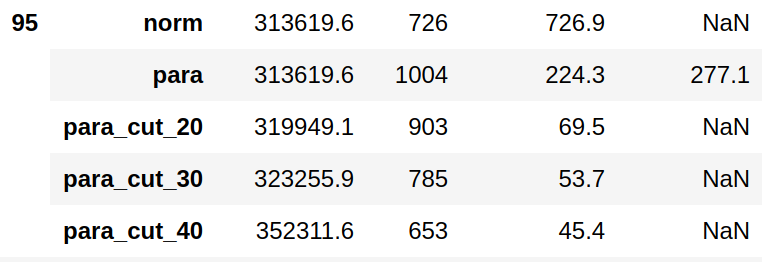


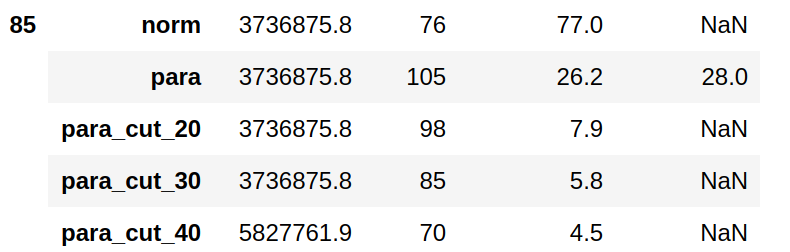
1. 对于cut来说，整体来看，是很有价值的，如下面的例子中，通过使用cut使得原本并行所需要的26.2s再次所见到了7.9s甚至cut\_30只需要5.8s，并且并没有降低算法的精确度。这说明剩下的30%的计算都没有价值了。下面两个表分别是规模为85和规模为100时，cut方法对原始方法对仅仅使用并行化来说都有非常显著的提升，且很低程度上影响精度。





1. 当然cut方法，也不总是很有效，有时候，也会一定程度上影响实验的结果如下面这种情况：当规模为95时，随着cut的线程的增多，其精度慢慢减少；另一情况，当规模为85时，cut为40时，其精确度降低相对来说，比较明显了。这取决与数据的分布，以及使用者对误差的允许范围





# Conclusion and Future Work

本次实验是关于分区问题的解法，分别给出了分区问题的背景和意义、具体的问题描述和解决分区问题的几种算法。重点展开讨论了ckk方法在分区问题具体实现。并对ckk方法并行化，并考虑在误差允许的范围内，提前终止并行分支。本文考虑的是针对如若70%的并行线程都结束了，则结束实验，返回minmax。

也可以考虑在50%的并行线程结束后，结束原来的并行，并重新开启32个新的线程进行下一步的计算，充分利用计算资源，使用资源的同时，保证了精度；或者，若已经得到允许的误差范围，则当运行到合适的误差范围后，提前终止并行，只是这样需要提前预判算法是否能够达到预期的误差范围中。其与本文的主要区别是，也是好处是，能够同时保证误差的稳定性，和算法的速度。

# 参考文献

[1] https://en.wikipedia.org/wiki/Partition\_problem

[2] Korf R E. A complete anytime algorithm for number partitioning[J]. Artificial Intelligence, 1998, 106(2): 181-203.

[3] Karmarkar N, Karp R M. The differencing method of set partitioning[M]. Berkeley: Computer Science Division (EECS), University of California, 1982.

[4] Korf R E. Multi-way number partitioning[C]//IJCAI. 2009, 9: 538-543.

[5] Schreiber E L, Korf R E, Moffitt M D. Optimal multi-way number partitioning[J]. Journal of the ACM (JACM), 2018, 65(4): 1-61.