Enhanced atom-by-atom assembly of arbitrary tweezers arrays方法解读

问题描述

•输入一系列原子的分布情况(origin)和目标分布(target)

•输出移动原子的操作方案，使总操作时间最短

注：每个原子均相同

分析

•使总操作时间最短，一方面要使所有原子移动总路程最短，另一方面还要减少移动次数（记为Nmv）

•考虑到各个算法的总移动路程差异不大，本文重点讨论如何减小（Nmv）

•影响最终结果的各个因素较为复杂，理论上的最优解往往无法真正达到，只能寻找到"较优解"

基本思路总述

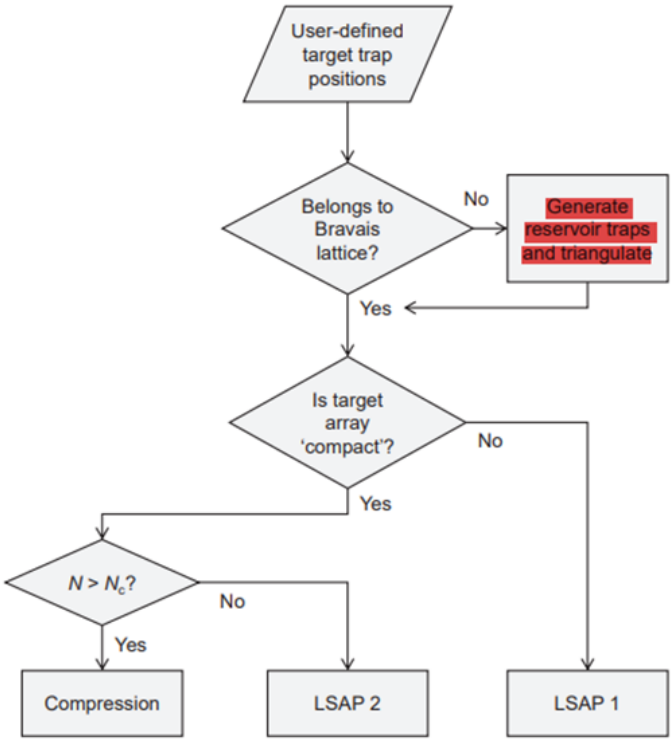


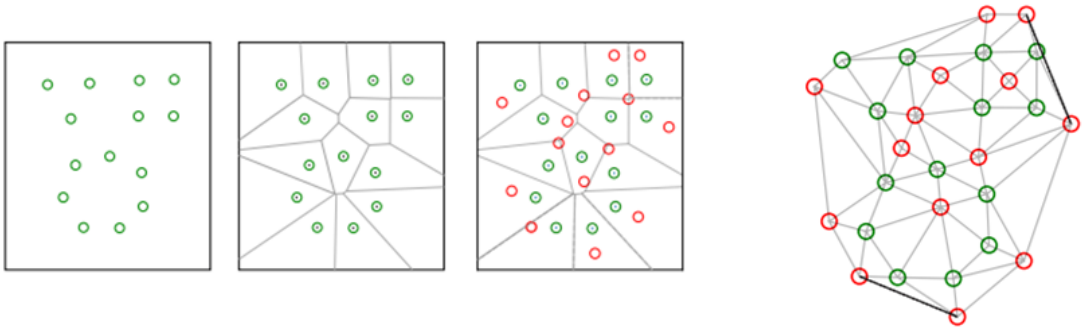
FIG. 8. **Algorithm choice flowchart.** The best-suited algorithm to be used depends on the characteristics of the target array.

基本思路：先导入原始(origin)和目标(target)的原子分布坐标列，再对导入数据进行分析，通过对其进行 Generate reservoir traps and triangulate 操作(后文会介绍)，将其转变为一个"图"（即数学图论中的"图"概念），节点表示可以放置原子的"trap"，边表示可移动的路径。如果目标排布不是紧密型的，则使用LSAP1算法；如果是紧密型排布，若总原子数大于Nc，则用Compression算法，否则用LSAP2算法。其中Nc~300，该算法选择方案为经验结论，即根据实际计算结果得出的最优方案，使Nmv最小。

实验预处理

方法：Generate reservoir traps and triangulate

目的：将输入数据(target)化为布拉维点阵，从而使问题转化为图问题

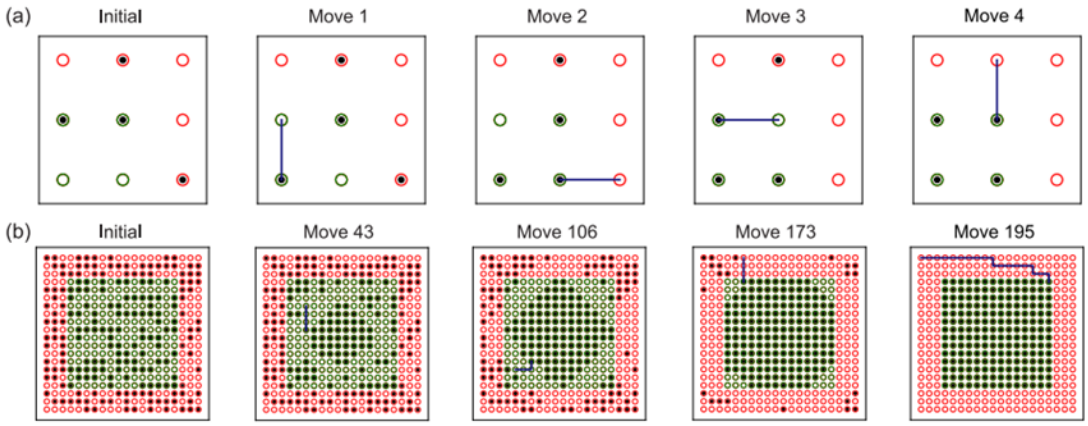


该部分的处理属于计算机图形学内容，具体做法较为专业，我还没有能力实现。

核心算法介绍

Compression

core：move layer by layer



黑点代表原子，绿圈代表目标位置

该方法适用于目标点阵紧密型排列的情况，思路较为自然：从目标排布的中央开始，一层一层向外扩展。对某一层的操作，即将最靠近该层的未归位原子拉到该层最近的空位（Shortest moves first算法）。通过逐层排布，最终实现目标排布。

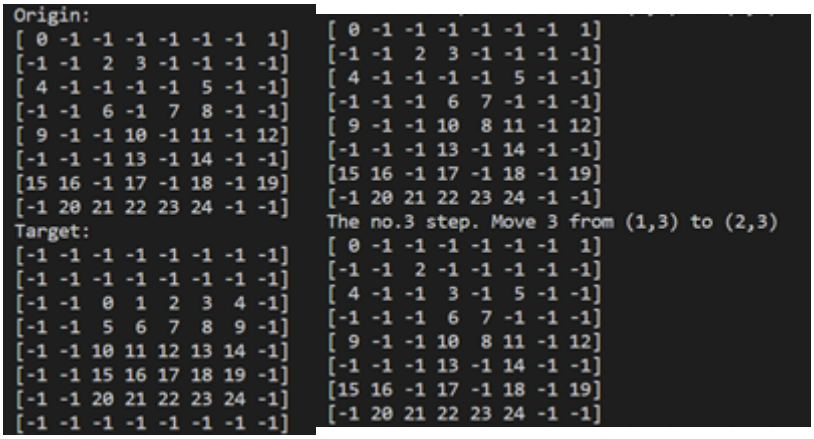
该算法的优点是思路简单自然，易于操作，且因为每次移动都是选择最近的原子，所以也就不存在某次移动需要经过其他原子的情况。总移动次数Nmv小于等于N。

代码实现

源代码见compression.py

输入样例见compression.in

输出结果 注：输出的演示矩阵中-1代表空穴，0，1，2，3，等数字代表原子编号，每一步的移动方式在输出结果中用 *Move {原子编号} from {起点坐标} to {终点坐标}* 表示



LSAP

此处先引入一个基本算法问题：Linear Sum Assignment Problem

基本概念：N 个人完成 N 件任务，已知第i个人完成第j项任务的花费为cost[i][j]，求出最优任务分配方式，使总花费最小

该问题在python中有现成的库，无需自己编写。

```
from scipy.optimize import linear_sum_assignment
```

LSAP1

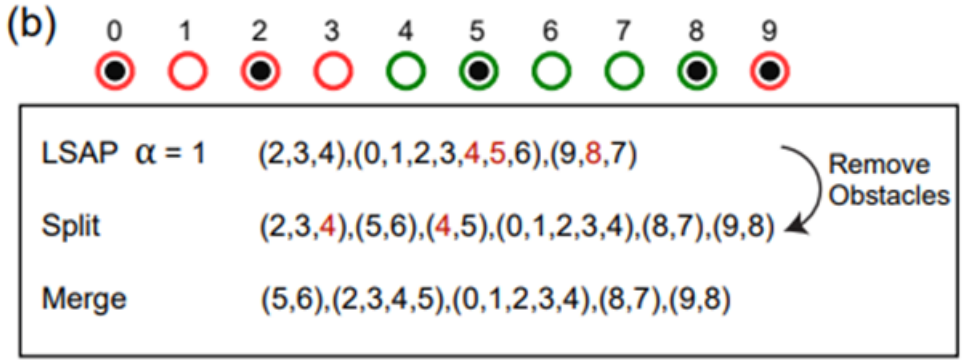
该方法适用于目标排布非紧密型排列的情况

step1 定义矩阵 cost[i][j]=distance[i][j]，其中distance指编号为 i 的原子到编号为 j 的目标空穴的曼哈顿距离

step2 对 cost 做 LSAP ，得出一个移动方案，即应将第 i 个原子移动到 第 j_i 个位置。在具体实现中每一次移动用一个列表表示，如[0,1,2,3,4,5] 表示将原本在0 位置的原子经过1，2，3，4，移动到5 位置的空穴处

step3 对操作过程进行分割迭代(split)。考虑到每次移动所经过的路径上可能有别的原子，如在移动操作：[0,1,2,3,4,5] 中2位置有别的原子，则将该操作分割为两次移动操作：[2,3,4,5], [0,1,2]

step4 二次合并迭代(second merging iteration)。合并可合并的移动，（以线状图为例）如图[2,3,4], [5,6],[4,5],[0,1,2,3,4],[8,7],[9,8] 中,[2,3,4]与[5,6] 操作之间并无影响，可交换两者顺序并将[2,3,4]与[4,5]合并，最终得出[5,6],[2,3,4,5],[0,1,2,3,4],[8,7],[9,8]共5次移动。



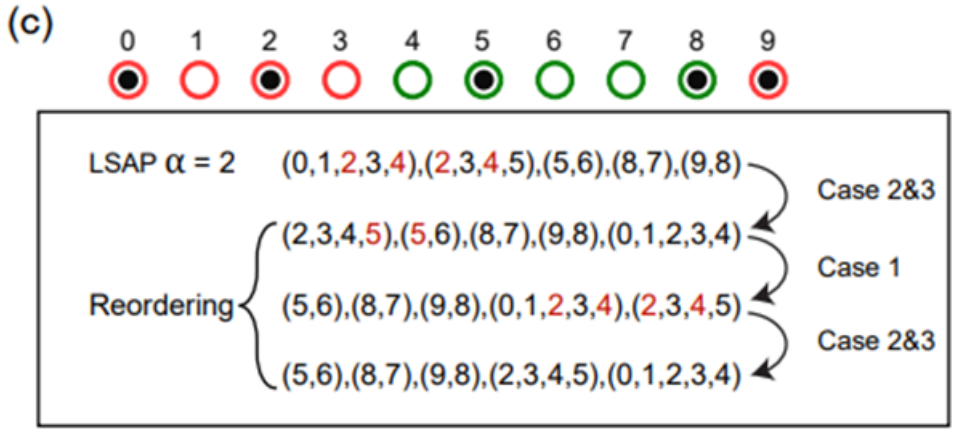
LSAP2

该法适用于原子数较少的紧密型排列

step1 定义矩阵 cost[i][j]=$(distance[i][j])^2$，其中distance指编号为 i 的原子到编号为 j 的目标空穴的曼哈顿距离。之所以用平方，是因为根据统计规律，综合考虑距离和路径中经过的其他原子，每次移动的总时耗大致与距离平方成正比。

step2 对cost矩阵做LSAP

step3 根据LSAP结果得出一系列移动操作，对这些操作进行重排。如果进行到的某个移动操作会经过其他原子，则将该操作置于末尾，先进行其他操作。最终得出不会造成原子间碰撞的操作序列



源代码：

见 LSAP.py

测试样例 见 LSAP2-1.py

三种方法比较

•Compression：适用于N>300的紧密型排布(target)

•LSAP2：适用于N<300的紧密型排布(target)

•LSAP1：适用于松散型排布(target)

注：以上结论均为经验结果

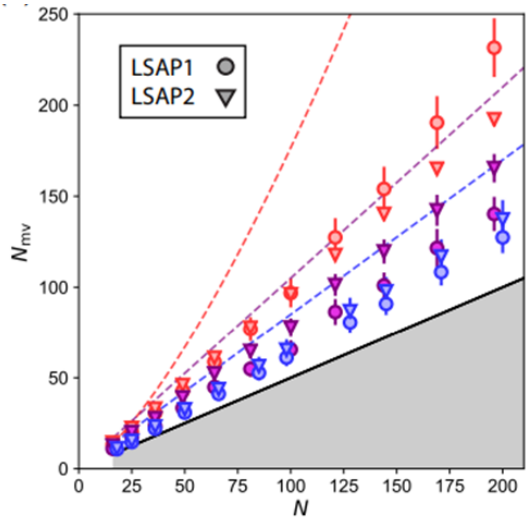


FIG. 5. **Modified LSAP algorithms.** (a) Using a cost function with $\alpha = 2$ (see text) in a LSAP solver favors short moves. (b) The algorithm LSAP1 first uses a LSAP solver with $\alpha = 1$, which returns a list of moves (here, $(2, 3, 4)$ means that the atom initially in trap 2 is moved, via trap 3, to trap 4). Some moves lead to collisions (denoted in red) and thus the set of moves is post-processed as in the "shortest-moves-first" algorithms, by splitting the problematic moves into two or more stages. However, in a second step, two moves that share the same trap as final and initial positions (denoted in red) can be merged together, reducing the total number of moves. (c) The algorithm LSAP2 uses a modified cost function with $\alpha = 2$, which returns a set of short moves; to avoid collisions, the moves are then reordered by applying successively three rules (see text) until the rearrangement can be performed without collisions. Numbers in red highlight the breaking of a rule. (d) Number N_{mv} of needed moves as a function of N to assemble a checkerboard subarray (blue), a random subarray (purple) or a compact subarray (red), for the LSAP1 and LSAP2 algorithms. The dashed lines reproduce the fits of Fig. 1 for comparison.

参考文献

Enhanced atom-by-atom assembly of arbitrary tweezer arrays, Kai-Niklas Schymik, Vincent Lienhard, Daniel Barredo, Pascal, Scholl, Hannah Williams, Antoine Browaeys, and Thierry Lahaye, Phys. Rev. A ***102****, 063107 – Published 10 December 2020*