人智大作业：七巧板拼图

2017011527 自 76 吴紫屹

2018 年 10 月 17 日

一、概述

本次人智大作业，我选择完成的是七巧板拼接的任务。使用 python3+pyqt5 构建图形化界面，两个必做任务的功能均有实现，同时选做 1 完成了 13 巧板、选做 2 完成了任意输入图像的拼接。

对于一个七巧板的谜题，程序基本上在几秒之内可以给出解决方案；对于 13 巧板问题，基本上在 1 分钟之内可以给出解决方案；对于任意图像输入的问题，基本上在 2 分钟之内可以给出解决方案。

在大报告的剩余部分，我将首先将此次大作业的题目建模成一个典型的搜索问题，然后叙述自己对于此类问题的一般性思路，接着针对实现的三种游戏模式具体阐述实现方法，最后则是效果的展示以及对于大作业的一些感想。关于代码结构、UI 使用方法的介绍则是在 README.md 中有详细的说明。

二、问题建模

本次大作业的主题为搜索，因此首先要将七巧板问题建模为一个标准的搜索问题，即定义出状态和状态之间的转换操作。为了叙述的方便下面我们首先约定一些名词的含义及其符号。

S(state)代表状态，F(function)代表状态之间的转移函数；M(mask)代表当前七巧板的待解图形，C(corner)代表几何图形的顶角，S(segment)代表几何图形的边，E(element)代表需要摆放的组件，例如七巧板中我们记大三角形(large triangle)、正方形(square)、平行四边形(parallelogram)、中三角形(medium triangle)、小三角形(small triangle)为LTr1, LTr2, Sq, Para, MTr, STr1, STr2。

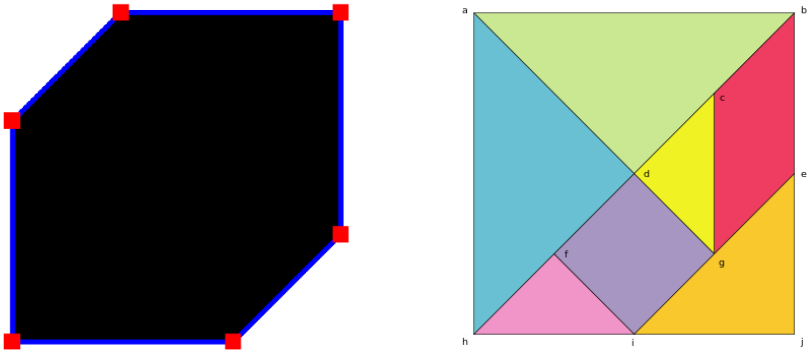

图 1 名词符号及其定义举例 1

例如我们看图 1 左，红色的点标注出来的就是程序检测出的顶角C，而蓝色线画出的就是边S，整个黑色的部分就是待求解的图形M。图一右则展示了典型的七块七巧板组件E。对于 13 巧板的E我们会在相应部分详述。

回到对于状态和转移操作的定义。我将状态S定义为当前待求解图形M和下一个要放置的组件E_{next}的集合：$S = \{M, E_{next}\}$；转移函数F定义为将E_{next}放置到M的合理位置上并且更新M。这里需要说明的是，对于三种游戏模式中，我都预定义好了一个固定的放置组件的顺序，例如在七巧板中是两个大三角形、正方形、平行四边形、中三角形、两个小三角形，这么设置的原因和好处都会在之后详细算法说明处阐述。

图 2 名词符号及其定义举例 2

同样举例以说明之。图 2 左展示了一个七巧板问题初始化后的M，此时的S为该M以及E_{next}即LTr1，通过转移函数F我们找到了一个合适的位置并将E_{next}成功放置上去(如图 2 中)，最后我们更新M得到图 2 右的效果。此时的S即为该M和E_{next}即LTr2。

不妨再以经典搜索问题的视角来看待七巧板拼接问题，经过我的定义后，问题成功建模：初始状态S_{init}由输入待求解图片直接转换而来，目标状态为拥有全白M(意味着完全填充)的S_{target}，程序所要实现的就是根据一个S的M决定E_{next}的摆放位置并更新S。

三、一般性思路

求解搜索问题的算法主要就是课上所讲述的几种：深度优先(以下记做 DFS)、广度优先(以下记做 BFS)和启发式搜索(以下记做 A*，虽然并不仅有这一种)。我将论述对于不同算法的优缺点及我的选择。

3.1 A*

最开始我尝试使用启发式搜索的算法来求解七巧板的问题，但是对于估值函数h(n)的选取一直没有什么好的思路。后来我意识到，七巧板问题和一般的适合于使用 A*算法的寻路问题有着极大的不同,这是由于对于给定的目标状态S_{target}，寻路问题中存在多种可行解，但是最优解可能只有一种，我们的目标是通过设计合适的启发函数来尽快地找到此最优解。但是在七巧板问题中，**所有可行解都是最优解**，我们只要尽快找到一个解就行，无须担心其是否最优。

另一方面，我认为在七巧板问题中如果要套用 A*的$f(n) = g(n) + h(n)$的公

式，那么一个直接的想法就是g(n)(已经付出的代价)是已经放置的组件数，而h(n)(还需付出的代价)是待放置的组件数，不难看出这样的定义对于我们搜索是没有任何帮助的(毕竟我们的操作只能是放置一个组件E)。而任何更为复杂的估值函数定义则实在不是三周大作业时间内可以发掘的，因此最后我放弃了 A*算法。

3.2 BFS

根据之前的分析，我们要做的是尽快找到一个解，而不担心其优劣性。因此BFS 是不适合这一任务的。

3.3 DFS

DFS 是我最终选择了的算法，因为它可以保证搜索的高效性。在七巧板问题中由于组件E总数是固定的，不会出现搜索到很深但是仍然得不到解的情况，同时也免除了人工剪枝的麻烦。因此 DFS 是一个简洁但是十分合适的选择。

3.4 一般性算法流程

使用一个栈Stack来实现 DFS。

1. 预处理待求解原始图片，初始化S_{init}，$\mathrm{Stack.push}(S_{init})$
2. 若Stack空，求解失败。否则，得到$S_{current} = Stack.pop()$
3. 使用F寻找所有合适位置在$S_{current}.\mathrm{M}$上放置$S_{current}.E_{next}$
4. 更新得到所有S_{next}，$\mathrm{Stack.push}(S_{next})$
5. 如果$S_{next} == S_{target}$，返回此S_{next}，搜索结束。否则，重复 2～5

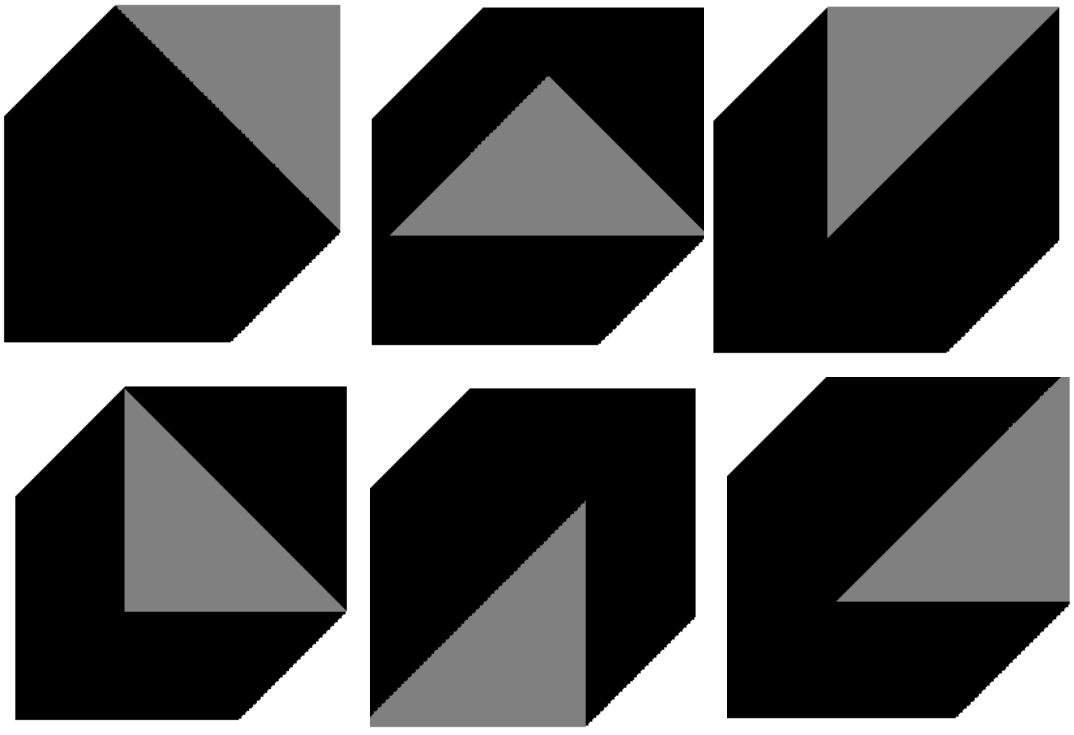

图 3 一般性流程举例

在图 3 中我可视化了一次搜索(仅截取部分结果)。对于$S_{current}$和E_{next}即LTr1，程序尝试在多个位置进行了合理的摆放(灰色位置)，得到的这些结果都将压入栈中(实际压入栈中的M不存在灰色的部分，灰色部分应也被涂成白色，此处仅仅是为了可视化的方便)。

3.5 对于搜索算法的最后一点讨论

虽然我的搜索算法是直接以 DFS 的形式予以实现，但实际上我认为也是包含了一些启发式先验知识在里面的。例如我之前提到的按照预定顺序放置组件E于M的合适位置，放置的顺序虽然是是直觉性地认为，大的图形比较不好放所以要先放，小三角形可以填补一些小地方所以要最后放，但是这其实相当于我把E_{next}的面积$S_{E_{next}}$当做了估值函数h(n)的一部分；同时，我每次都是放置在合适的位置，这相当于我认为任何不合理的放置位置的代价为+∞。因此我认为或许我实际上是使用了 A*算法来指导我的搜索，只不过代价函数h(n)长得比较奇怪：

$$\mathrm{h(n)} = I_{(invalid\ element\ position)} * (+\infty) - S_{E_{next}}$$

其中$I_{(invalid\ element\ position)}$在下标的条件成立时取 1，否则取 0。

四、详细算法

接下来我将针对三种游戏模式对于算法进行细致的解释。每一部分都包括M和E的具体定义、$I_{(invalid\ element\ position)}$的判断、基于先验知识的E搜索方法。

4.1 经典七巧板

M定义：将待求解图片二值化，置需要放置E的区域值为 0(黑色)，置不能放置E的区域值为 255(白色)，就得到了M。值得注意的是这里我没有对输入原图大小进行任何变换大小(resize)之类的操作，最大程度避免了非整数(non-integer)坐标近似(如$\sqrt{2}$近似为 1)造成的误差。

E定义：对于不同组件，我实现了三个类(Triangle、Square、Parallelogram)，为了表征M上具体放置的一个组件，E拥有其顶角C的坐标、使用两个有坐标的端点来表征的边S等成员变量。这些都是为了之后$I_{(invalid\ element\ position)}$的判断。

$I_{(invalid\ element\ position)}$判断方法：一开始考虑的方法是，在一个和M相同形状(shape)的矩阵中将E_{next}放置区域置 1，与M直接相乘求和，如果E_{next}位置不合理则上述结果将大于 0(因为M中不能放置的区域值为 255，能放置的区域值为 0)。但是试验之后发现速度极慢不可行。遂考虑使用基于C和S的判断方法。

具体而言，对于E_{next}，我们拥有其C和S的具体坐标，对于M，我同样使用 opencv 检测其C和S的坐标，因此可根据以下三步进行判断：

1. E_{next}的所有C需要在M的内部，通过坐标很容易实现
2. E_{next}的所有S和M的所有S不能相交，通过向量的外积实现
3. E_{next}的所有S和所有E_{used}(放置过的组件)的所有S不能相交

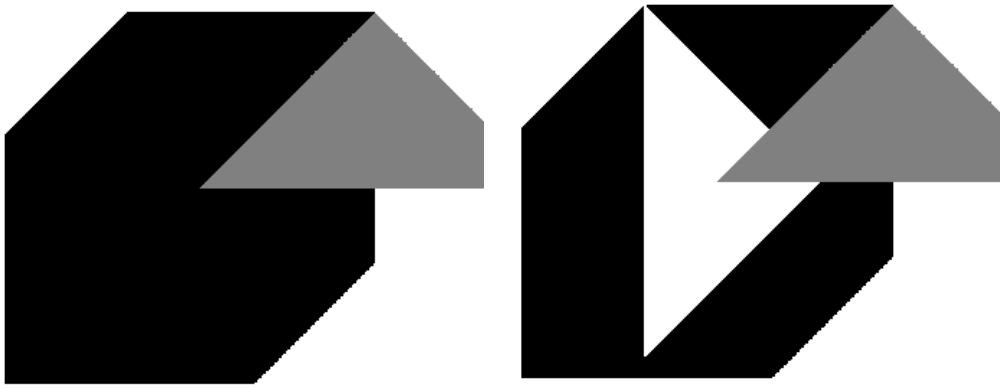

图 4 不合理摆放举例

如图 4 是两个典型的不合理摆放样例(灰色部分为E_{next})。左图中E_{next}的右下顶角C显然在M外，且其斜边与M的一条边相交；右图中E_{next}的一条直角边与之前摆放过的LTr1(图中白色大三角形区域)的一条直角边相交。

通过将E抽象成C和S的集合，我们只需要极少的计算量和比较次数就可以判断任意形状几何图形之间的关系，大大加速了判断过程，同时也更加准确。

E搜索方法：在七巧板问题中，我不是逐像素搜索可能的E摆放位置，而是根据M的顶角C进行。这是由于我发现在所有情况下，LTr1、LTr2不可能都被其余组件所完全包围，至少会有一个LTr的C和M的C重合(通过边长计算容易证明)。因此我就可以只遍历当前M的所有C，进行$I_{(invalid\ element\ position)}$的判定。

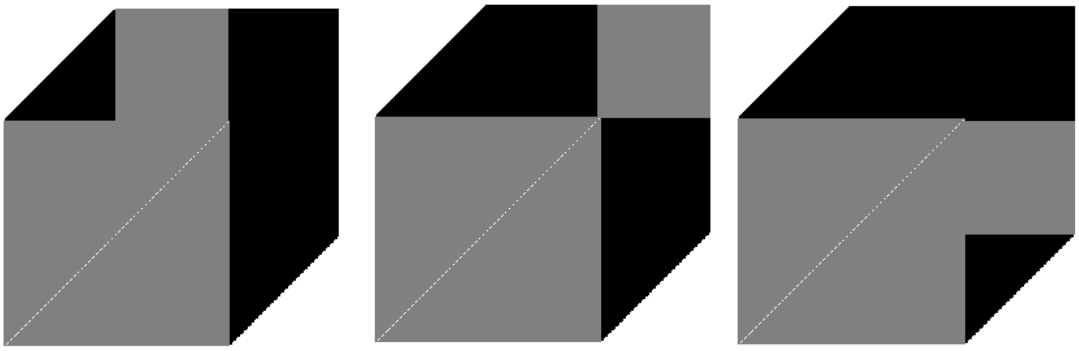

图 5 E搜索方法举例

同样举例说明，在图 5 中我们已经摆放了LTr1、LTr2，在对于Sq的尝试中，可以看到只选取了三个C的位置进行摆放，很快就得到了三种可能的摆放位置。

总而言之，对于经典七巧板问题，我将E抽象成C和S的集合，得以快速判断位置是否可以摆放，同时使用了遍历M的C的搜索方法，大大减小了搜索空间，提高了搜索的速度和准确率。

4.2 13 巧板

13 巧板的E如下图 6 所示：

图 6 13 巧板组件

13 巧板的代码设计中我使用了很多的先验知识：

1. E都可以拆解成1x1小正方形的组合
2. E的面积都是 5，只有一个正方形组件的面积为 4

M定义：由于 13 巧板E都是“方方正正”的缘故，这里的M不再是待求解的原图，而是将原图经过网格化处理后的栅格(grid)。这意味着，一个1x1小正方形直接对应于栅格M中的一个像素。这十分有利于$I_{(invalid\ element\ position)}$的判定。

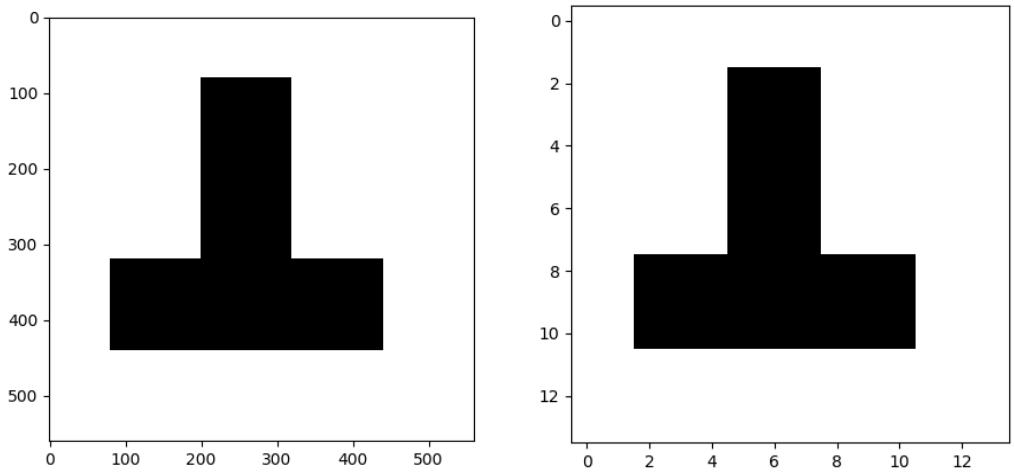

图 7 栅格化举例

如图 7，注意横纵坐标的数值，左图为大小为560x560的原图，而右图为大小为14x14的栅格M，这样做减小了搜索空间且没有损失任何精度。

E定义：E的定义比七巧板中的简单很多，只需要有面积S_E个点的坐标即可。

$I_{(invalid\ element\ position)}$**判断方法：**由于栅格化操作使得M中一个像素相当于一个1x1小正方形，因此我们只需要验证一个E中所有坐标的点对应M中像素值都

为 0 即可。

E搜索方法：在 13 巧板中任何E都可能被其他E所包围，且需要注意的是很多 13 巧板问题并需要用上所有的E，往往只需要其中的 9 或 10 个，因此我们无法按照M的C来进行搜索，只能逐像素进行。幸运的是，栅格化操作大大减少了M的像素个数且$I_{(invalid\ element\ position)}$极其简单，因此也不会很慢。

此外，为了进一步缩小搜索空间，我会根据输入图片面积判断是否需要正方形(唯一一个面积为 4 的E)，如果需要的话就第一个放置。那么对于此后的每一个M，其每一个连通域的面积必然都是 5 的倍数！通过这一准则我们可以剪掉许多不可能成功的放置方法。

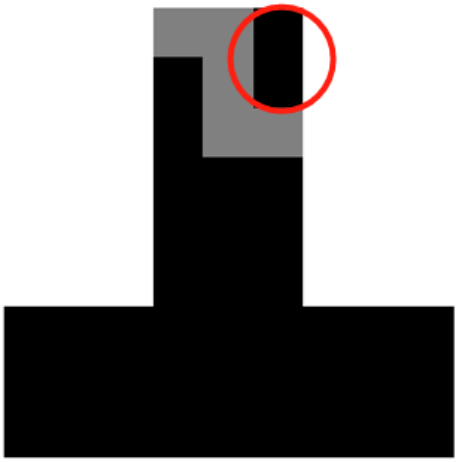

图 8 连通域面积错误举例

如图 8 所示，右上角部分的面积仅为 2，显然不可能被任何E所填充，因此这一情形是不合理的。

总而言之，13 巧板问题中使用了极为重要的先验知识，即将E分解为1x1的小正方形和M的栅格化处理，实现了快速的搜索和求解。

4.3 任意输入图形七巧板

M定义：与经典七巧板类似仍是原图，但是由于任意图形一般都是不规则的，因此我们无法检测其C和S。

E定义：与经典七巧板相同。

$I_{(invalid\ element\ position)}$**判断方法：**由于为了近似一个不规则的图形，E是极有可能伸出M边界的。因此与经典七巧板问题相比，我松弛了E_{next}与M的冲突判断，但是严格了E_{next}与E_{used}的冲突判断。具体实现是通过一些改变阈值。

E搜索方法：由于不规则图形不存在C，我们无法像经典七巧板问题中那样使用遍历C的方法。因此在这里我们只能使用逐像素搜索。处于搜索空间大小的考虑，我在初始化S_{init}时对输入原图进行了缩小(resize)的操作，使得不同图片其求解时间基本相同。此外，针对任意输入问题我还进行了如下的特殊处理：

1. 引入计算机视觉(computer vision)领域中常用的IoU(交比并)来度量搜索到的拼接结果的质量，定义为$\text{IoU} = \frac{S_{all\ elements} \cap S_M}{S_{all\ elements} \cup S_M}$，越接近 1 越好
2. 不期待搜索得到的结果可以实现 100%的IoU，而是手动设置一个IoU数值(如 0.8)，根据这个系数来计算图片的比例尺(unit_length)的值
3. 为了进一步减小逐像素搜索的复杂度，我并不是真的每个像素遍历，而是对不同大小的E定义了不同的遍历间隔。基于的假设是仅仅使用规则E拼接得到的几何图形，那些E_{large}(如LTr)必然是会伸出M的，因此在M内部进行这些E_{large}的摆放尝试是没有意义的；而对于那些E_{small}(如 STr)，它们可以在M的内部填补空缺，增加IoU，所以对于它们的遍历间隔就要比较小，以得到精细的结果。总之就是采用了一个粗粒度和细粒度结合的思路。
4. 由于对于一个不规则图形，在我所给的限制条件下可能存在多种可行解，而这些解中存在优劣关系，因此程序不是直接返回第一个搜索到的解，而是返回前K(如 10)个可行解中IoU最大的一个当做最优解
5. (未实现)就人类的视觉而言，IoU最大的图实际上并不一定看起来最“神似”原图。这是由于一些细节部分如下图 9 中狗的四肢、尾巴，它们很纤细因此对于IoU的贡献不大，极容易被程序忽视，但是它们对于视觉相似度的影响却是巨大的。为了解决这一问题我考虑过使用 A*算法寻找一个代价函数，将纤细部分的IoU赋予更高的权重来平衡这一问题，但是由于代码实现上的困难最终放弃了。这或许是将来改进的一个方面。

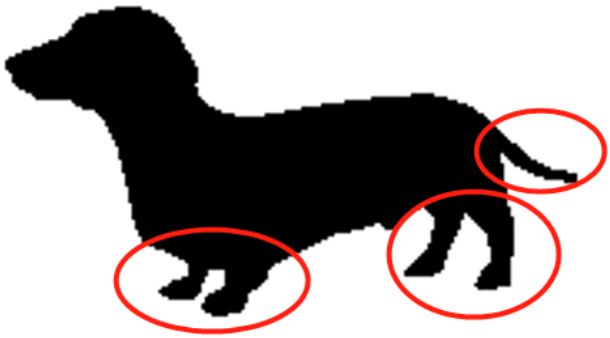

图 9 任意输入图像举例

总而言之，对于任意图像输入的七巧板问题，我通过松弛和严格部分约束条件，并且引入IoU指导的方法，能够搜索到具有一定相似度的拼接方案，但是仍然有很大的改进空间。

五、结果展示

本节主要展示几幅程序求解出来的拼图。同时我也会放上一些定量的准度和精度的度量指标。

5.1 经典七巧板

经典七巧板数据集中总共收集了 57 个问题，主要可以分为两大类：一类是具有明确物理意义的物体，例如图 10 中 1.png 是一个奔跑的人；另一类则是一些抽象的平面几何图形，例如图 10 中 24.png 是一个六边形。总的来说，对于第一类的拼接是较为简单的，因为其形状鲜明，空间足够大能放得下LTr的地方有限，在此基础上搜索也就快得多。而第二类由于形状普通，合理不出界的放置位置很多，所以搜索起来会慢一些(甚至一个数量级)。

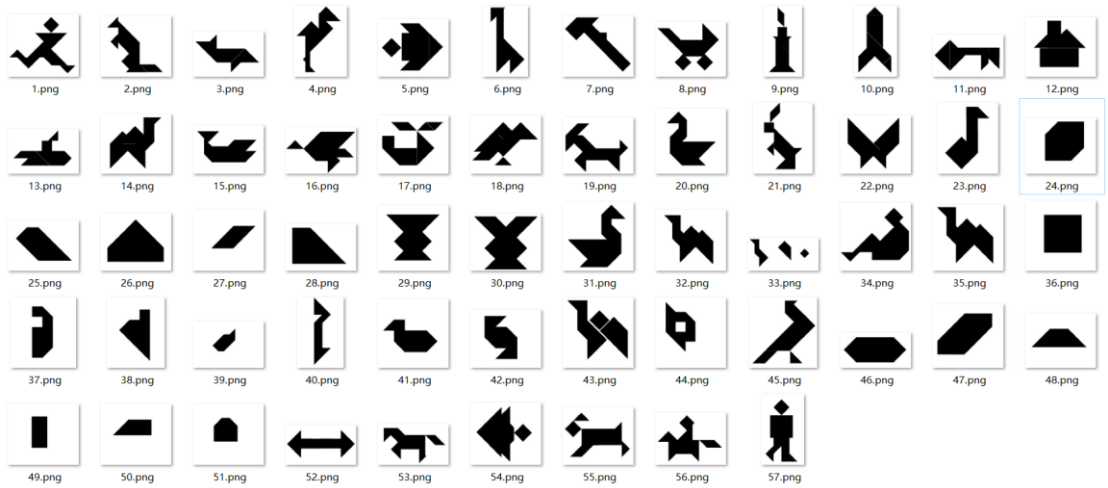

图 10 经典七巧板问题举例

	第一类图形	第二类图形	所有图形
平均搜索时间(s)	1.263	8.463	5.684
平均搜索次数(次)	2810.8	18893.6	12686.2

表 1 经典七巧板问题定量分析

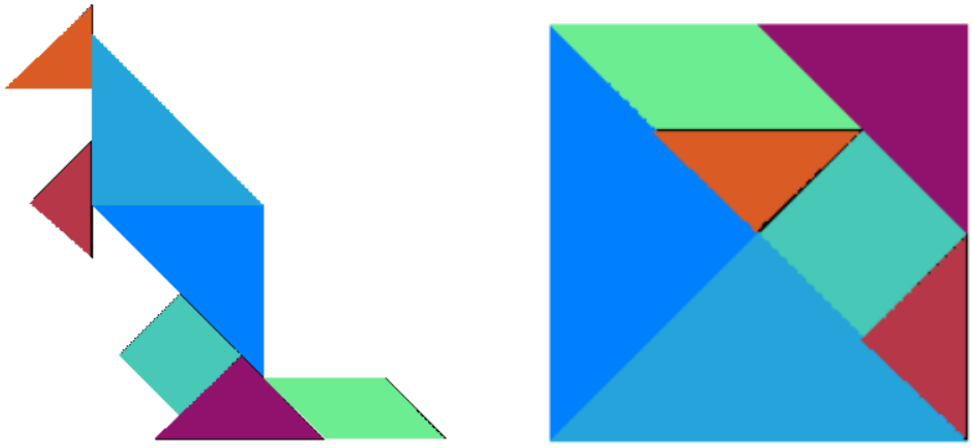

图 11 经典七巧板两类典型问题举例

如图 11 左侧为一个典型的第一类图形，搜索时间为 1.36s，搜索次数为 4395 次；右侧是一个典型的第二类图形，搜索时间为 9.43s，搜索次数为 29364 次。可见确实存在极大的不同。

另外，如果在此类问题中使用暴力的逐像素搜索法，假设M的边长为10^2量级，

那么其搜索次数应当为$(10^2 \times 10^2 \times 8)^5$(每个E(LTr和STr都只算一个)都可能在每个像素放置，且每个像素可能有 8 种角度旋转)，这个数量级是不敢想象的。因此我的一些同学采用了先将图片缩小(resize)到较小尺寸再逐像素搜的方法，但是这样由于像素插值的问题可能出现较大误差(尤其锯齿状边缘损害极大)。这也证明了我所采用的基于C的搜索方式成功减小了搜索复杂度。

5.2 13 巧板

13 巧板由于网上的数据集资源较少且图片质量极差，故而自己生成了 4 个问题。定量的分析及样例如下所示：

图 12 13 巧板问题举例

	搜索时间(s)	搜索次数(次)	搜索时间*(min)	搜索次数*(次)
1. png	110. 83	1027821	>60(太慢放弃)	N/A
2. png	47. 44	288066	46	$\mathbf{1.7 \times 10^7}$
3. png	36. 54	222521	33	$\mathbf{1.1 \times 10^7}$
4. png	23. 29	208061	21	$\mathbf{8.2 \times 10^6}$

表 2 13 巧板问题定量分析

加‘*’表示未加连通域面积判定的情形

由于 13 巧板E的数量比七巧板多出几乎一倍，同时采用逐像素的搜索方法，所以花费时间和次数明显多于前一个任务。另外我们可以看到，我所加上的基于先验知识的连通域面积判定大大较少了搜索复杂度(两个数量级！)，这也证明了课上讲到的，通过引入启发式信息可以帮助程序更快更准地搜索。

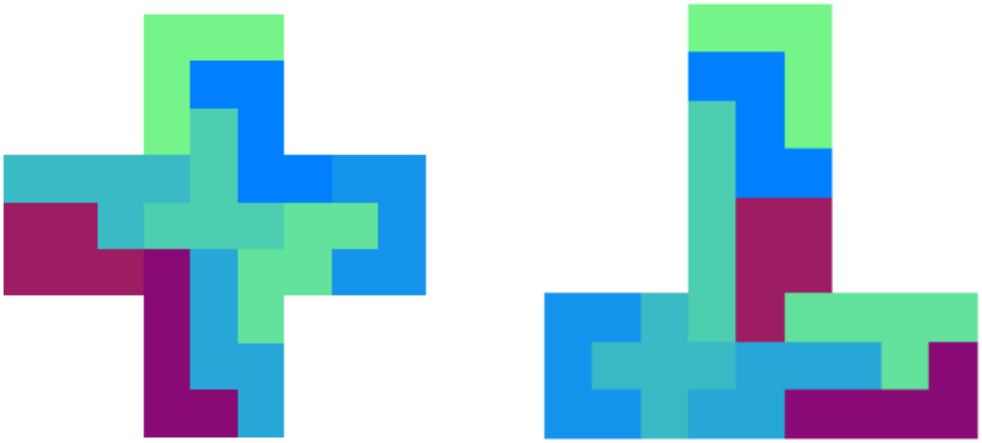

图 13 13 巧板拼接结果举例

5.3 任意输入七巧板

此任务我的完成情况一般，虽然投入了大量时间设计算法但是最终效果总是差强人意。由于这一问题中我们搜索得到的可行解往往不是最优解，因此搜索速度和次数的指标没有什么意义，IoU是一个比较科学的考量方式(个人感觉IoU和搜索时间的关系有点类似于 CV 领域的P(precision)R(recall)曲线，我也是试了几组问题找了一个比较合适的超参数来平衡两者的取舍)。

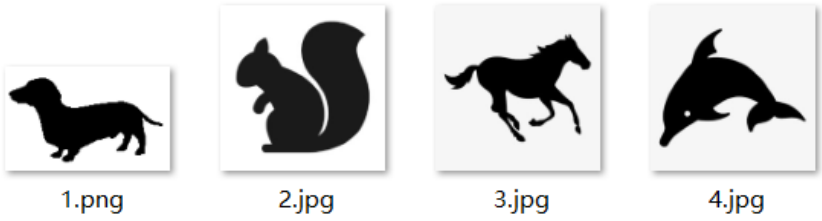

图 14 任意输入七巧板问题举例

图 15 任意输入七巧板结果举例

例如在大作业样例中给出的狗的图形，搜索得到如图 15 的IoU = 0.72的结果，花费时间为 80 秒左右，但是如果要得到我的程序可行的最优解IoU = 0.78的结果，我尝试过在实验室的服务器上跑了一晚上花了大概 3 个小时才跑出来，应当来说这个取舍还是很不值得的。

	IoU(%)	搜索时间(s)
1. png	72. 1	83. 5
2. jpg	72. 4	301. 2
3. jpg	62. 3	56. 1
4. jpg	65. 0	15. 95

表 3 任意输入七巧板问题定量分析

总之对于这一任务最终实现的效果确实一般，可以作为未来的TODO项。例如，相比于设计一个可以着重考虑四肢等边角纤细部位的估值函数h(n)，更简单的方法应当是对原始输入图片进行预处理，手动将M中较细的部分加粗，这样在计算IoU时就会占更大的比重。另外我也看过有关改进IoU的指标，使得新的指标更符合人体视觉系统，可惜由于时间有限无法尝试了。

六、收获与总结

本次人智大作业以搜索为主题，当时看到两个题目时我毫不犹豫就选择了七巧板，因为我觉得这一任务更为有趣，同时也极富挑战性。通过三周的工作，我成功地一个抽象的拼图问题建模成了一个典型的搜索问题，从而可以使用人工智能发展史上一些经典的搜索算法进行求解。刚拿到题目的我其实是完全没有思路的，但是当我强制自己定义出状态S、转移函数F等变量之后，顿时就豁然开朗，不禁感慨前人智慧确实让后人获益无穷。

第二个感触比较深的地方是先验知识的重要性。如果用暴力穷举的方法解决的话，那搜索空间将是惊人的大，但是通过任务一中遍历C的方法、任务二中栅格化的操作、任务三中不同粗细粒度搜索间隔，我成功减小了搜索的时间和次数。我认为经典人工智能和一般程序最大的不同就在于引入了对于问题的启发式知识，而不仅仅在盲目求解。（不过说句题外话，现在深度学习这么火爆，好像什么先验都不需要网络自己学就能学得很好，我本人对此还是比较怀疑的）

第三点则是认识到了建模思维的重要性。比如说将M视为C和S的集合，而非一个原始的二维矩阵，这对于各种操作都方便得多。

最后一点是在写报告过程中，我充分意识到将编程是脑中的思路转化成文字是多么的困难。为了达到这一目标我尽量使用符号、公式和图表进行说明，但还是有事词不达意，望助教海涵。

总而言之，此次大作业在难度上、训练价值上我认为都是十分合理的，这三周下来本人获益匪浅。