2020 秋人工智能基础第一次大作业报告

基于 A*算法的重排 M*N 宫问题

2018011702 自 84 孙浩文

目录

1 完成任务简述

- 通过基本 A*算法实现原始的重排九宫问题；
- 实现了较为方便且容错性强的可视化 UI 界面。
- 支持随机生成与手动点击修改来实现初始九宫的排列；
- 以 A*算法及其修改升级版本实现了 M*N 宫的重排。

2 问题描述与算法设计

2.1 问题描述

重排九宫的基本问题为：给出 3*3 的九宫格，其中一个位置为空白，其余 8 个位置上填入 1-8 数码，玩家可以将空白块与其上下左右的数字进行交换，目的是通过尽量少的步骤实现数字排列位置与目标一致。

与此同时，该问题还可以升级为 M*N 版本，即在长为 M、宽为 N 的“MN 宫格”中设置一个位置空白，其余位置上填入 1 至 MN-1 的数码，其余规则一致。

为了使得 UI 界面的简捷与操作的方便性，本次大作业中的最终排列目标均为正序排列，即在 MN 宫格中从左到右、而后从上到下分别为 1，2，...，MN-1，在右下角方格中为空白块。最终排列目标示意图如下：

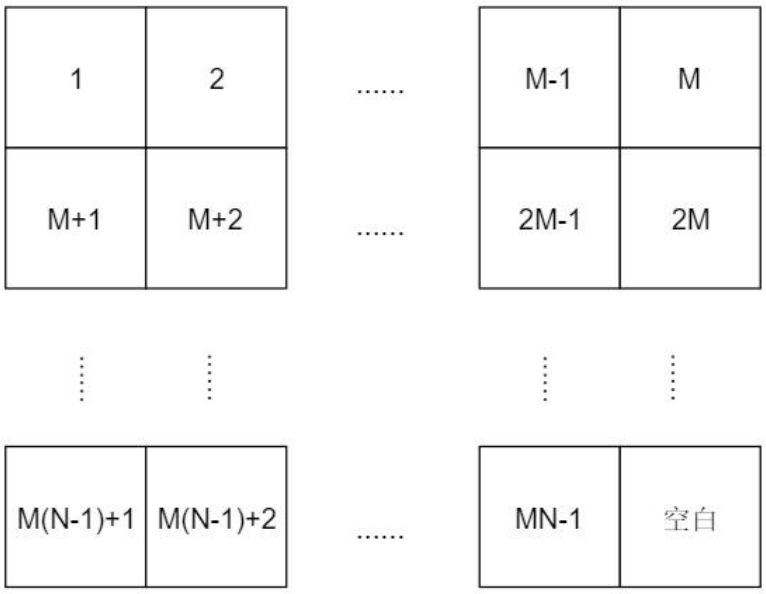


图 1 最终排列目标示意图

2.2 算法设计

2.2.1 传统 A*算法

由于重排九宫是一个较为经典的搜索问题，因此本次大作业中使用的最为基本的算法即为 A*算法。对于任意一个节点 n，其对应着一种状态，定义 A*算法中的代价函数如下：

$$f^*(n) = g^*(n) + h^*(n)$$

其中 $g^*(n)$ 代表当前节点的深度，即为从开始状态到当前节点对应状态移动的步数，而 $h^*(n)$ 代表当前节点对应状态与目标状态之间各元素曼哈顿距离之和。与课上所提及的推荐方式不同，这里并没有加入逆序数对的计算，原因是对于每种状态计算逆序数对总是需要进行二重循环，花费的时间不亚于曼哈顿距离的计算，但是收效甚微，因此为了节约时间成本，这里仅采用曼哈顿距离作为 $h^*(n)$。

上述方案在进行初始重排九宫（即 3*3 宫）上有着优秀的表现，但一旦进行到稍稍高一些的阶数上表现就不尽人意。因此，在上述基础上重新定义代价函数如下：

$$f^*(n) = g^*(n) + k(w,h)h^*(n)$$

$k(w,h)$ 为根据当前方格阵长宽尺寸设置的一个经验系数。根据本人的尝试，经验系数定为：

$$k(w,h) = 6wh$$

由于随着问题规模的扩大，步数的增长速度远快于曼哈顿距离和的增长速度，若不进行系数的设置，则该搜索问题容易近似为一个宽度优先搜索，从而导致时间较长。而在规定了上述系数后，整体的时间都有了明显的下降，且步骤数并没有明显的增长。对于 3*4、4*4 基本可以做到较为快速的求解。

2.2.2 速度更快的 A*算法——open 表剪枝

即便采用了上述的较为合理的代价函数，对于更高阶数的问题，往往还是需要非常多的求解时间，因此我把注意力放在了算法的其余部分。我发现随着问题规模的增长，open 表的增长速度过大，导致每次进行搜索时往往会在 open 表中存入过多的节点，而对于那些代价值过大的节点，我们不需要使用却总是存储并遍历，这是较为多余的步骤。

而通过实验我发现，即便是对于 5*5 宫格问题，长度为 25 的 open 表也足以

支持其搜索出结果。因此我对于每一次向 open 表中加入节点时进行判断：如果当前 open 表中的节点个数超过了 25，则去除掉表中代价值最大的节点，以保证表中节点数的稳定。

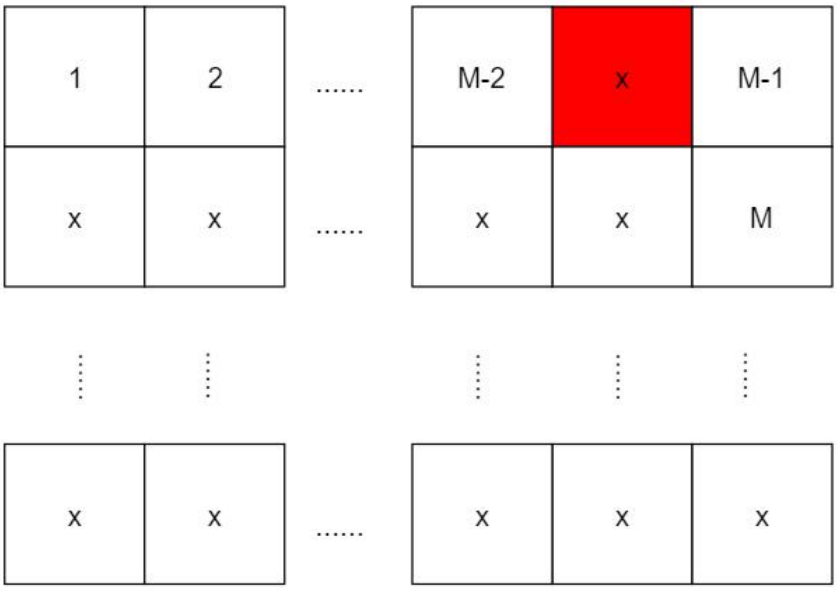
通过测试，上述方法能够完成 5*5 及以下的所有问题，并且耗时较短（具体耗时测试请参考第 4 部分）。不过这么做的代价便是完成搜索后的移动步数略有增加。

2.2.3 实现速度步骤双保障——基于分治策略的半 A*搜索

上述两种主体方法虽然已经能够实现 5*5 问题的求解，但是针对于更高阶的问题却稍显乏力。与此同时，时间上的快速和步骤上的简洁似乎存在着不可调和的矛盾，我希望能够获得一种更为迅速、对更高阶问题有良好表现且步骤数较为合理的方法。

基于上述想法，我将自己代入到这个游戏中，想象我在完成 M*N 宫、且目标为图 1（第一页），则我往往是从上到下，先完成第一行的排布，在进行第二行、第三行，并且通过前两个算法在较高阶情况下的搜索结果也得到了类似的结论。因此，我重新构建出了一种基于分治策略的半 A*算法。

首先，对于任意 M*N（宽为 M，高为 N）的问题，若最顶上一排已经完成了从 1 至 M 的正序排列，那么问题即可简化为一个（M-1）*N 的问题，而对于最顶层一排的排列，可以考虑如下目标形式：



其中，前 M-2 个数均处于对应位置上，而 M-1 在 M 的目标位置上，M 在 M 的目标位置下方，x 处的数码或白块任意，这样，只需要让白块移动至图中标红处，再将白块进行右—下两步移动，即可完成最顶层一排的构建。

为了达到上述的目的，我们先对于靠左的数进行排布，即排布顺序为从 1 至 M-2，首先定位到该数的当前位置(temp_pos)，而后通过白块的移动使得该数码与白块左右贴近，如果该数的目标位置(dest_pos)在当前位置的左侧则贴左边，右侧则贴右边，而后交换白块与当前数码，这样可以使得白块在水平方向上靠近目标位置。需要注意到的是，白块在行走时需要尽量靠近右侧和下侧，除非 temp_pos 正好处于右边界或者下边界，这样可以保证不对已排好部分进行破坏。

在上述移动过程中，对于 temp_pos 不处于右边界或者下边界的情况，只要白块不处于当前位置正上方，都先上下移动至 temp_pos 下面一行，而后左右移动对准需要的贴边列，最后上移一格即可。若处于正上方，则进行右移一格后再进行上述操作，示例图如图 2 所示；对于下边界，则先上下移动至倒数第二行，而后左右移动至需要的贴边列，再下移一格即可；对于右边界而言，直接左移至倒数第二列，然后上下移动即可。

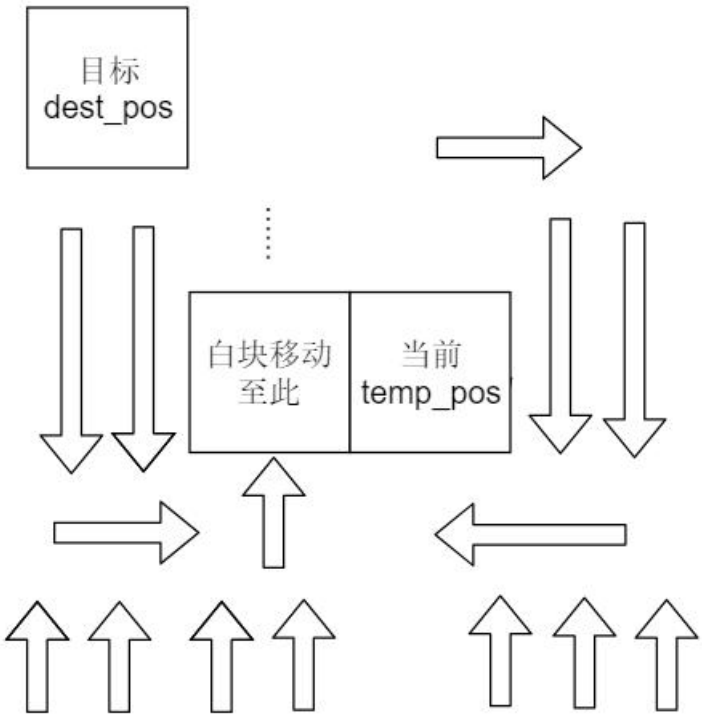


图 2 水平方向调整示意图

在水平方向上完成排列以后，白块需要通过从右侧绕到 temp_pos 的上方。此时，只要白块不是在 temp_pos 的同一列，均通过上升或下降的方式到达 temp_pos 的下一行，而后水平移动到 temp_pos 的右一列，最后移动至目标位置（示意图如图 3），这样可以保证不触碰到已经完成排列的数字。

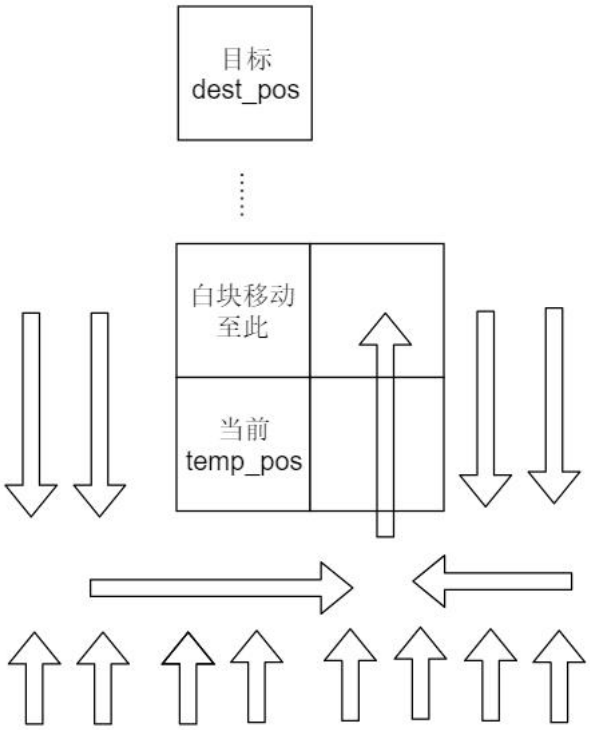
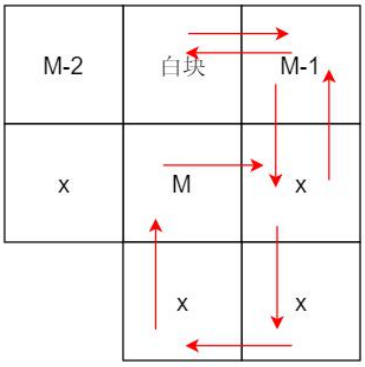


图 3 垂直方向调整示意图

当然在完成 M-1 的排布后，需要防止如下情况的出现：

1	2	.....	M-2	白块	M-1
x	x	.....	x	M	x

如出现以上情况，则按照下图箭头方向走一圈即可：



如此继续，即可完成目标型的转变，从而完成一行的排布。可以证明，对于 N-2 行及前面的行，均可通过如上的方式进行排列。最后剩下的 2 行，即为 M*2 宫格，则通过之前的 A*算法进行排布。因此最终，该方法将原本的 M*N 宫格通过不断的分治，最终变为了一个 M*2 宫格，因此大大降低了计算量。与此同时，使用本方法能够更为有效地控制步骤量。根据命令行中的测试，该方法可以在几秒钟内完成 7*7 宫格的排布。但是考虑到 UI 界面的显示效果，在 UI 部分仅展示 5*5，高阶测试可以使用代码文件夹下的 test_Algorithm_DC.py 进行测试。

3 UI 界面

3.1 基本界面展示

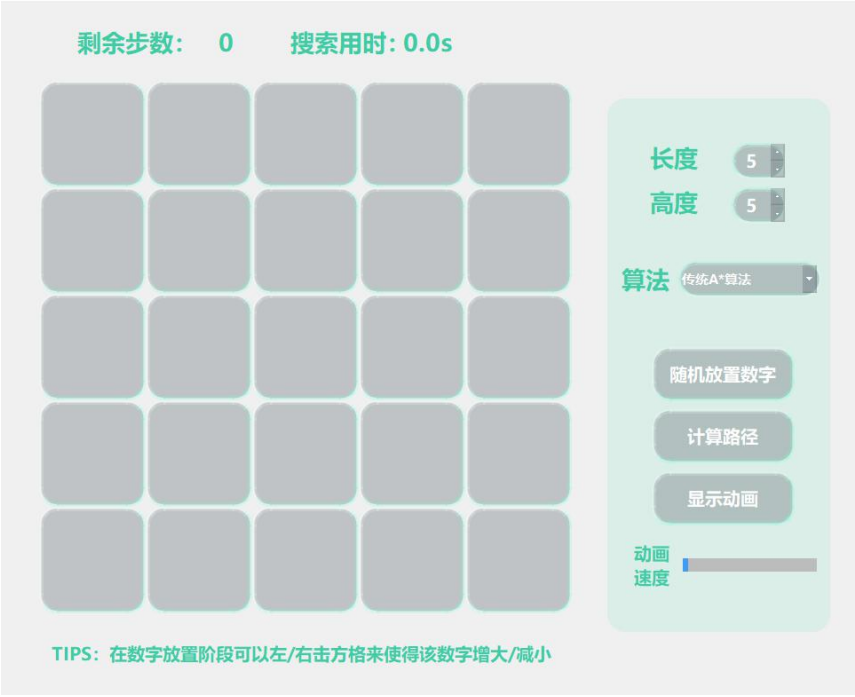


图 4 基本 UI 界面图

在进入界面后，有如下可以供调节的选项：

• 长度、高度：直接进行 MN 宫大小的调节，范围各自从 3 至 5，优先级最高，可以打断动画演示，一旦调节则展示出调节后的大小，并呈现出空白状态，此时可以左/右击 MN 宫格任何一个方块来增大/减小数字，也可以点击“随机放置数字”按钮来随机出题；

• 算法选项框：可以下拉并从 3 种算法中选择一种，分别对应于文中 3 种算法描述。其中传统 A*建议在 4*4 及以下操作（预期时间最多半分钟，不排除意外情况），更为快速的 A*在 5*5 时可能会近似 1 分钟；基于分治的 A*则均为秒解（具体还可参考第 4 部分的时间）；

• 随机放置数字按钮：进行基于当前大小的随机出题，题目未必可解，由点击“计算路径”按钮后判断，在路径计算完成后以及演示动画期间即便按下也无法调整；

• 宫格方块：鼠标左/右击分别可以增大/减小数码，实现手动出题；

• 计算路径按钮：通过逆序数对判断数字放置是否正确（有无重复），如无则先判断是否有解，若有解则根据当前的算法方式进行计算路径，完成后会在最上方显示出步数以及搜索时间；

• 显示动画按钮：在完成计算路径后点击即可展示动画；

• 速度调节滑块：移动滑块可以调节动画速度。

3.2 操作方法

在进入界面后，首先进行长宽的选择，同时选择希望使用的算法，而后可以出题，既可以直接点击宫格进行手动出题，也可以点击“随机放置数字”按钮进行自动出题，之后点击“计算路径”按钮，若排布不符合规范（有重复数字或空白）或者题目无解则会弹窗提醒并可以重新调节题目，题目有解则进入搜索，完成后会在最上方显示出步数以及搜索时间，最后点击“显示动画”按钮即可播放对应动画，与此同时可以调节速度，或者直接调节大小来打断动画播放，进行新的出题。

4 时间统计

为方便判断时间，我进行了 20 次随机求平均值的方式来测试各种算法对于各种大小的平均时间，记录如下表所示（下一页，单位：秒）。

表 1 不同算法对于不同规模问题的平均时间

问题规模	传统 A*	速度更快的 A*	基于分治的 A*
3*3	0.007591	0.0490	0.01369
3*4	1.113	0.3616	0.04416
4*4	15.38	4.451	0.05190
4*5	-	21.98	0.1198
5*5	-	55.19	0.1290
6*6	-	-	0.6585
7*7	-	-	5.015

由于是平均时间，因此可能存在一定差异性，但是整体而言可将上表作为参考。关于 6*6 与 7*7 的测试，可以运行 test_Algorithm_DC.py 进行测试。

5 总结与收获

本次大作业中，基于重排 MN 宫问题，我以 A*算法为出发点，完成了多种算法的尝试、实现与优化，最终最优算法达到了至少“秒解”5*5 宫格的程度。与此同时，我对 A*算法的原理和具体实现机制有了进一步的理解，同时也对 PyQt 的 UI 界面设计有了进一步的认识，可谓收获满满。