Lab1 实验报告

- PB17111568
- 郭雨轩

数码问题

启发式

本实验采用的是`Manhattan + Linear Conflict`作为启发式，下面分别论述其正确性：

- `Manhattan` 可以看作原问题的松弛问题需要移动的步数，对应的松弛问题为：假定不管想要移动的块的相邻位置是否为空，都进行移动。基于此，启发式自然是可采纳的。
- `Linear conflict` 是在`Manhattan` 启发式上的改进。具体体现为，当前状态中，若两个块处在同一行或同一列，且在目的状态中也处在同一行和同一列，且没有归位，这时则出现线性冲突，物块之间不仅需要`Manhattan` 启发式计算出的移动，还需要在水平/竖直方向上进行额外的移动。这个启发式我参考了这篇论文[1]和博客[2]，论文也放在了实验报告的目录下面，其中有关于这个启发式的证明，在这里不再赘述。

算法分析

我的A^* 和 IDA^* 是严格的按照书上的算法进行实现的，仅仅将IDA^* 的实现改为递归的形式

A^*伪代码如下，时间复杂度为$O(b^d)$，d取决于启发式函数：

```
function reconstruct_path(cameFrom, current)
    total_path := {current}
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.prepend(current)
    return total_path

// A* finds a path from start to goal.
// h is the heuristic function. h(n) estimates the cost to reach goal from node n.
function A_Star(start, goal, h)
    // The set of discovered nodes that may need to be (re-)expanded.
    // Initially, only the start node is known.
    // This is usually implemented as a min-heap or priority queue rather than a hash-set.
    openSet := {start}
```

```

    // For node n, cameFrom[n] is the node immediately preceding it on the cheapest path from start
    // to n currently known.
    cameFrom := an empty map

    // For node n, gScore[n] is the cost of the cheapest path from start to n currently known.
    gScore := map with default value of Infinity
    gScore[start] := 0

    // For node n, fScore[n] := gScore[n] + h(n). fScore[n] represents our current best guess as to
    // how short a path from start to finish can be if it goes through n.
    fScore := map with default value of Infinity
    fScore[start] := h(start)

    while openSet is not empty
        // This operation can occur in O(1) time if openSet is a min-heap or a priority queue
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        for each neighbor of current
            // d(current,neighbor) is the weight of the edge from current to neighbor
            // tentative_gScore is the distance from start to the neighbor through current
            tentative_gScore := gScore[current] + d(current, neighbor)
            if tentative_gScore < gScore[neighbor]
                // This path to neighbor is better than any previous one. Record it!
                cameFrom[neighbor] := current
                gScore[neighbor] := tentative_gScore
                fScore[neighbor] := gScore[neighbor] + h(neighbor)
                if neighbor not in openSet
                    openSet.add(neighbor)

    // Open set is empty but goal was never reached
    return failure
```

IDA^*伪代码如下，时间复杂度为$O(b^d)$，d取决于启发式函数：

```
root=initial node;
```

```
Goal=final node;
function IDA*()                                                   //Driver function
{
    threshold=heuristic(Start);
    while(1)             //run for infinity
    {
      integer temp=search(Start,0,threshold); //function search(node,g score,threshold)
        if(temp==FOUND)                                  //if goal found
            return FOUND;
    if(temp== ∞)                               //Threshold larger than maximum possible f value
            return;                               //or set Time limit exceeded
      threshold=temp;

  }

}
function search(node, g, threshold)              //recursive function
{
    f=g+heuristic(node);
    if(f>threshold)             //greater f encountered
      return f;
    if(node==Goal)               //Goal node found
      return FOUND;
    integer min=MAX_INT;     //min= Minimum integer
    foreach(tempnode in nextnodes(node))
    {
        //recursive call with next node as current node for depth search
        integer temp=search(tempnode,g+cost(node,tempnode),threshold);
        if(temp==FOUND)            //if goal found
          return FOUND;
        if(temp<min)     //find the minimum of all ‘f’ greater than threshold encountered
          min=temp;
    }
    return min;  //return the minimum ‘f’ encountered greater than threshold

}
function nextnodes(node)
{
  return list of all possible next nodes from node;
}
```

实验结果

本实验中，三个问题我均得到了最优解，分别为24步，12步，57步。

对于前两个输入，A^* 和 IDA^* 都仅仅需要很短的时间就可以得到精确解，且IDA^*的时间和内存消耗都更优，对于第三个输入样例，我在一台内存为`128GiB`的服务器上仅对A^*算法进行了测试，在25分钟内得到了精确解，内存消耗约为`100GiB`，由于一些原因（服务器是嫖的AWS且账户余额不够了），导致未能对IDA^*算法进行测试，不过可以预期的是，IDA^*要比A^*有更好的性能。

运行代码可以参照`README.md`。

使用`time`命令对输入测量时间：

	A^*	IDA^*
input1	time: 0.016s, step: 230	time: 0.078s, step: 62169
input2	time: 0.015s, step: 30	time: 0.017s, step: 27
input3	time: 25min, step: about 2.1 Billion	N/A

数独问题

算法分析

- MRV：每次选择变量时选择值域最小的变量
- 度启发式：每次选择具有最多约束的变量（数独中行、列、宫格、对角线上空位最多的变量）
- 前向检验：每次进行尝试赋值后，缩减当前所有变量的值域

使用的优化手段：MRV，前向检验，度启发式结合

实验结果

同样使用`time`命令进行时间测量（时间很少时会存在误差，请以step作为衡量相对性能的指标）

	input1	input2	input3
无优化	time: 0.024s, step: 111	time: 0.027s, step: 14853	time: 2.59s, step: 4233934
forward checking	time: 0.035s, step: 57	time: 0.025s, step: 2424	time: 0.287s, step: 197676
MRV	time: 0.025s, step: 88	time: 0.021s, step: 1813	time: 0.664s, step: 891146
MRV + forward checking	time: 0.020s, step: 47	time: 0.027s, step: 102	time 0.028s, step: 3475
MRV + degree + forward checking	**time: 0.020s, step: 47**	**time: 0.019s, step: 88**	**time: 0.027s, step: 579**

可以看到，当三种方式联合使用时，步数最小，效果最好。

思考题

- 可以使用遗传算法。
  - 初始化化种群
    首先需要产生较优的初始种群，以减少进化代数，如果没有较优的初始种群会加大后面运算压力尽可能使填入的数字与所在行或列的数字不重复。根据以上规则得到一定数量的初始九宫格，然后将每个方格缺的数字按从上到下、从左到右的顺序连在一次作为染色体。
  - 交叉
    将染色体随机两两组合，随机取两个染色体中间相同的位置进行交换，交叉完后，将未交叉的重复元素用另一个染色体的重复的元素交换（因为该染色体重复的元素就是另一个染色体缺少的元素，元素守恒）。
  - 变异
    按变异率在种群中随机选择一定数量的个体，随机产生一个变异节点（一一个九宫格的方格作为一个节点），将该节点左右翻转。
  - 选择
    将父代、子代、变异代三部分染色体合在一起，计算每个染色体还原到九宫格中行和列重复数字的个数，初始分为8*（9+9）=144，每重复一次减去一分。选出分数最高一部分作为下一轮进化的父代。进化到一定程度，出现分数等于144时，退出进化。
- 可以使用爬山算法和模拟退火算法。先对问题进行转化，通过赋予数独一个“能量”，能量计算的规则是：同一个九宫格，同一行，同一列任何两个数字如果一样那么能量就是1，如果不一样那么能量就是0。只能当总能量为0的时候，此时能量最低，而且满足数独完成条件。通过给与数独一个能量的概念和计算规则，我们将数独问题转换成一个寻找最低能量问题，对于这个问题使用爬山算法或者模拟退火算法进行求解即可。