Lab 1 Report

PB20111623 马子睿

A*算法求解L型锁问题

问题描述

- 对于一个$N \times N$的01矩阵，每次可以拨动L型的3个元素使其取反。求出使矩阵中全部元素全为0的最少步数及其对应方案

评估函数的选择

- 在本题中，我选择了这样的一个评估函数：

  令$D_1, D_2, \ldots, D_m$为矩阵中所有的**1值8连通片**（1值8连通片：以上、下、左、右、上左、上右、下左、下右八个方向为1而定义的连通所构成的连通片），其中每个8连通片中的1的个数为$h_1, h_2, \ldots, h_m$。

  $$h = \sum_{i=1}^{n} \lceil \frac{h_i}{3} \rceil$$

  用简单的话来讲，这里的评估函数就等于**每个1值8联通片中的1值数量除以3取上整的和**

- **可采纳性证明**：

  先来证明一个定理：

  > 定理1：对于任意形状的8连通区域，若其中1的个数为n，则取$h = \lceil \frac{n}{3} \rceil$是可采纳的

  - 证明：

    1. 若$n\%3 = 0$，则显然最优的情况就是每三个点组成一个L型，因此这时最少步数肯定为$\lceil \frac{n}{3} \rceil$，也就证明了$h \leq h^*$
    2. 若$n\%3 = 1$或$n\%3 = 2$，则考虑最理想情况，则$\lfloor \frac{n}{3} \rfloor$步后肯定还剩余1了（这是因为$\lfloor \frac{n}{3} \rfloor \times 3 < n$，那么至少还需要1步来消除所有1，也就是至少需要$\lceil \frac{n}{3} \rceil$步解决，也就证明了$h \leq h^*$

    综上所述，定理得证。

  在这个定理基础上，我们再来证明：

  > 定理2：包含任意n个不同的1值8连通片的矩阵，$\sum_{i=1}^{n} \lceil \frac{h_i}{3} \rceil$依然是可采纳的。

  - 证明：

    所有的8-联通片都不可能和其他的8-连通片有翻转交互（也就是一次翻转反转了两个8连通片内的元素），否则这两个连通片一定是8-连通，也就不可能是两个分离的连通片。因此，对于每个连通片而言，定理1都成立，再由上述论证，定理2也成立。

  综上所述，我们证明了我们的启发式函数式可采纳的。

- **一致性证明**：

  最坏情况下，任何一次翻转至多将总的1个数减少3，而且这3个1只可能来自于同一个连通片（因为一次翻转不可能同时改变两个1值8连通片内的1），因此如果从N状态1步变为N'状态的话，开销是1，但由上述讨论可知，$h(N) \leq 1 + h(N')$，故这个启发式函数是一致的。

算法主要思路

- 数据结构：

  我使用了一个multiset来维护优先队列，每一个节点的结构如下：

  ```
  struct node {
      int f{0}, g{0}, h{0}, cnt{0};
      std::bitset<144>mat;
      std::vector<path_node> path;
  };
  ```

  每一个节点维护了自己的f，g，h和1的个数cnt，以及一个用bitset表示的布尔矩阵和一条通往该节点的路径。

- 算法描述：
  - 初始化：根据输入创建一个根节点，计算其f和h值，并将其插入优先队列中
  - 从队列中弹出一个节点，找到这个节点的矩阵中的一个1，将其沿12个方向进行展开（覆盖这个1的L有12种形态），创建新的状态节点，并计算其各项成员变量，之后插入优先队列。如果这个节点的矩阵和队列中已经存在的某个状态的矩阵相同，那么抛弃掉f值较小的状态节点。
  - 如果队头节点的h值变为0（在Dijkstra算法中应该检查盘面是否全为0），那么证明找到了解，算法结束，否则返回上一步。

实验结果记录与分析

在本实验中，A*算法一定能找出最优解，且没有空间受限；而为了保护电脑，我对Dijkstra算法使用了空间受限的搜索算法。

各项测试的总时间和弹出队列的状态数记录如下：

A*算法

	总用时/ms	总步数	弹出状态数
input0	2	5	107
input1	0	4	53
input2	0	5	51
input3	3	7	174
input4	0	7	40
input5	2	7	130
input6	102	11	3237
input7	36	14	931
input8	49	16	1208
input9	1298	23	32131

Dijkstra算法

	总用时/ms	总步数	弹出状态数
input0	3	5	389
input1	0	4	78
input2	2	5	277
input3	28	7	4043
input4	32	7	5035
input5	24	7	3577
input6	38995	323	3068692
input7	时间过长	/	/
input8	时间过长	/	/
input9	时间过长	/	/

可以看出，由于Dijkstra并没有对于未来的估计，导致了拓展节点数量爆炸性增长，搜索的时间也是大幅提高。而得益于未来的估计，A*算法可以只拓展那些最有可能达到目的状态的节点，这回使其错误尝试的次数大幅下降，从而完全超越了Dijkstra算法。

CSP求解工作分配问题

问题描述

- 制作排班表，需要满足以下条件：
  - 同一个工人不能连续工作两个时间段
  - 每一个工人至少要分到$\lfloor \frac{M}{N} \rfloor$个工作，其中M为工作总数
  - 尽可能最大化工人请求

CSP基本信息

- 变量集合：$X = \{W_1, W2, \dots, W_M\}$，表示有M份工作
- 取值集合：每个变量的取值集合$D_i = \{P_1, P_2, \dots P_N\}$，表示一共有N个工人
- 约束集合：
  - 相邻工作不能由同一个工人完成：
    $C_1 = \{\forall i \in [1, M-1], W_i \neq W_{i+1}\}$
  - 每一个工人至少要分到$\lfloor \frac{M}{N} \rfloor$个工作
    $C_2 = \{\forall i \in [1, N], \sum_{j=1}^{M} I(W_j = P_i) \geq \lfloor \frac{M}{N} \rfloor\}$

  $C = C_1 \cup C_2$

算法主要思路

- 从所有工作中找到一个没有被分配工人，且被请求数最少但不为0的工作，若没有则找一个被请求数为0的工作，设为w
- 如果没有找到这样的工作，那么证明所有的工作已经都被分配，这时需要比较当前工作是否优于之前的最优方案，如果更优，那么更新最优方案，同时如果这个方案已经满足了所有的请求，那么证明这个方案一定是全局最优解（即满足请求数=总任务数-无人请求的任务数），返回true，否则返回false
- 从这个工作的可能取值中寻找一个**有请求且当前被分配工作最少的工人。**如果不是所有工人都已经被分配了最少要求的工作的话，那么如果这个工人已经被分配了最少要求的工作，则重新选一个工人，否则为这个工人安排工作w；如果所有工人都已经被分配了最少要求的工作，那么直接为这个工人安排工作w
- 安排工作后，更新相邻左右工作的可能取值（这里我通过宏定义方便地实现了是否要更新两侧的请求列表），并递归调用这个函数。
- 如果递归的结果是true，那么代表算法找到了一个符合题意的全局最优解，直接返回true，否则需要恢复邻居的可能取值（和请求列表），并回到第3步

优化方法

- **最少剩余值启发式**

  在本实验中，我们需要找最优解而不是可行解（事实上找可行解会变得非常简单），因此我们也应当对MRV进行少许修改，使其变为**最少请求值启发式**，也就是对应了算法思路中的第一步：寻找一个请求数最少但不为0的未分配工作。这种启发式在"易于满足可行且最优"的输入情况下的效果非常好，因为我们算法的最终目的还是找到一个最优解，在请求数较多时最少请求启发式很容易就可以找到一个最优且可行解，而最少剩余值启发式只能很快找到一个可行解

- **前向检验**

  在代码中，我在每次递归前，都会通过当前分配的节点来更新左右节点的可行值集合，如此一来就避免了在分配完成时对方案进行限制检验。

- **最少约束值启发式**

  通过分析题目可以知道，本次实验分配的最重要部分还是在于平均分配的部分，因为可以自由分配的部分只占很小的一部分。而这时，我们采用了优先有意愿且最少已分配工作的工人的方法来进行取值选择，就是为了能更好的为其他工作留下更多选择，因为如果我们只按照意愿分配，那么很容易出现有一个工人已经分配满了，导致有些工作无人可干的情况出现。

- **提前剪枝**

  在算法运行过程中，如果发现一个已经分配工作的方案中，未被满足的请求个数已经大于等于总工作数减去当前最优方案的成功满足请求数，那么就不再进行进一步搜索，因为进一步搜索只可能带来更多的未被请求工作。

实验数据记录

在这里我们引入一个统计量：平均错误搜索深度。这个搜索深度将会在发生回溯时更新。这个平均错误搜索深度越接近工作总数，证明算法就越没有产生大范围的回溯（因为如果产生了大回溯，那么回溯时的深度就会较小，这个值就会较小）。如果平均错误搜索深度无穷，证明没有任何一次搜索时需要回溯的。

	总用时/ms	总满足数量/最大满足数量	平均错误搜索深度
input0	0	20/21	17.3
input1	0	60/60	∞
input2	0	33/33	∞

	总用时/ms	总满足数量/最大满足数量	平均错误搜索深度
input3	0	114/114	∞
input4	0	69/69	60.6062
input5	13	576/576	∞
input6	34	1008/1008	∞
input7	4	378/378	∞
input8	102	2160/2160	∞
input9	37	720/720	653.44

可以看出，综合了上面几种优化的算法效果非常不错，很多搜索都是不需要回溯的，而需要回溯的算法其平均错误搜索深度也都接近最大深度，证明搜索回溯并不剧烈。

input 0 方案展示

```
1,2,1
2,3,1
3,2,1
3,1,2
3,2,1
3,2,3
2,1,3
20
```

观察input0可以发现，第12号工作根本无人申请，其他工作均有人申请，故12号工作无法满足，其他工作都可以满足。