启发式搜索

Wanjun Zhong Nov.7

内容

- 理论课内容回顾
  - 启发式搜索
  - A*
  - A*的性质
  - IDA*
  - 启发式函数设计
- 实验课任务与报告提交

内容

- 理论课内容回顾
  - 启发式搜索
  - A*
  - IDA*
  - 启发式函数设计
- 实验课任务与报告提交

启发式搜索

启发式搜索又叫有信息的搜索，它利用问题所拥有的启发信息来引导搜索，达到减少搜索范围，降低问题复杂度的目的。

- 无信息搜索对所有的可能路径节点一视同仁，而启发式搜索可以指导搜索向最有希望的方向前进
- 如何评估一个节点的重要性？ - 估价函数

$$f(x) = h(x) + g(x)$$

其中g(x)是从初始节点到节点x付出的实际代价；而h(x) 是从节点x到目标节点的最优路径的估计代价。h(x)建模了启发式搜索问题中的启发信息，是算法的关键。启发式函数的设计非常重要，合理的定义才能让搜索算法找到一个最优的问题解。

A* 搜索算法

A*算法可以看作是BFS算法的升级版，在原有的BFS算法的基础上加入了启发式信息。

- A*算法的估价函数也是 $f(x)=h(x)+g(x)$
- 算法描述

从起始节点开始，不断查询周围可到达节点的状态并计算它们的f(x),h(x)与g(x)的值，选取估价函数f(x)最小的节点进行下一步扩展，并同时更新已经被访问过的节点的g(x)，直到找到目标节点。

- 算法优缺点：

拥有BFS速度较快的优点，但是因为它要维护"开启列表"以及"关闭列表"，并且需要反复查询状态。因此它的空间复杂度是指数级的。

A* 搜索算法

- 算法步骤

- **开始**

1. 从起点开始，把其当成待处理点存入一个“开启列表”。

2. 搜寻起点周围可能通过的节点，也把它们加入开启列表，为这些节点计算f(x),g(x),h(x)，并且将节点A存为“父节点”。

3. 从开启列表中删除节点A，将其加入“关闭列表”（列表中保存所有不需要再次检查的节点）

- **循环直到找到目标节点或者“开启列表”为空（无路径）**

4. 从“开启列表”中找到估价函数值最低的节点C，并将它从“开启列表”中删除，添加到“关闭列表”中。

5. 检查C所有相邻节点，将其加入“开启列表”，将C作为它们的父节点。

6. 如果新的相邻节点已经在“开启列表”，则更新它们的g(x)值

A* 搜索算法

→ step cost = 200

→ step cost = 100

h(n1) =50

h(n3) =50

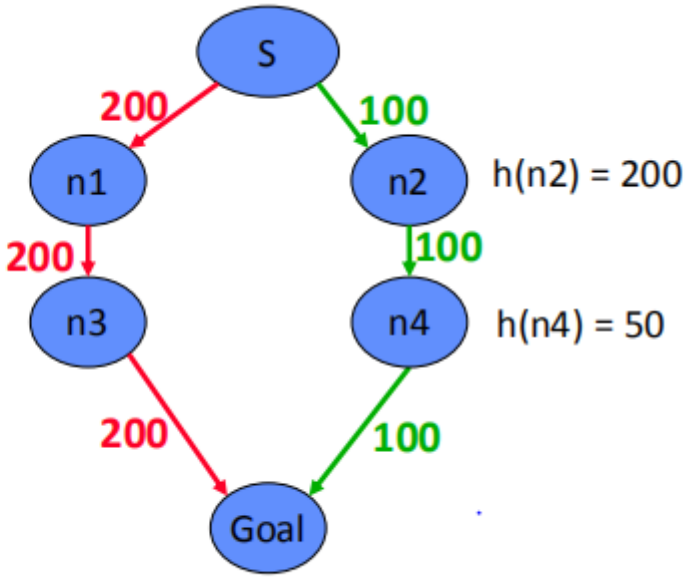
g(n)+h(n)=f(n)

{S} → {n1 [200+50=**250**], n2 [200+100=**300**]}

→ {n2 [100+200=**300**], n3 [400+50=**450**]}

→ {n4 [200+50=**250**], n3 [400+50=**450**]}

→ {goal [300+0=**300**], n3 [400+50=**450**]}



IDA*（迭代加深A*）搜索算法

IDA* **是迭代加深深度优先搜索算法（**IDS）**的扩展。因为它不需要去维护表，因此它的空间复杂度远远小于**A*。在搜索图为稀疏有向图的时候，它的性能会比A*更好。

- IDA***算法的估价函数也是** $f(x)=h(x)+g(x)$
- **算法描述**

在算法迭代的每一步，IDA*都进行深度优先搜索，在某一步所有可访问节点对应的最小可估价函数值大于某个给定的阈值的时候，将会剪枝。

- **算法优点**

当问题要求空间复杂度比较低的时候，IDA*更有优势。

IDA* 搜索算法

- 算法步骤

对于给定的阈值bound，定义递归过程

1. 从开始节点C，计算所有邻居节点的估价函数，选取估价函数最小的节点作为下一个访问节点

2. 对于某个节点，如果估价函数大于阈值，则返回当前节点的估值函数值。

3. 对于某个节点，如果是目标节点，则返回状态"到达"

注：初始阈值一般设置为起始节点到目标节点的估价函数值。每次迭代增加阈值为超过之前阈值的最小节点估价函数值。

启发式函数设计

- 启发式函数h(n)告诉算法从任何节点到目标节点的最小代价估计值，它的选取很大程度影响算法性能。

h(n) 的值	描述	性能变化
h(n) = 0	只有g(n)起作用，退化为Dijkstra算法	保证找到最短路径
h(n) <= h*(n)		满足单调性时，保证能找到最短路径
h(n) = h*(n)	只遵循最佳路径不会扩展其它节点	运行速度快并且能找到最短路径
h(n) > h*(n)		不能保证找到最短路径

启发式函数设计

- 性质1：可采纳的（admissible）

当估价函数的预估值小于等于真实值时，并且函数满足单调性时，算法必然可以找到一条从起始节点到最终节点的最短路径。这种性质叫做可采纳的。

$$h(n) \le h^*(n)$$

- 性质2： 单调的 （consistent）

当节点n的估价函数值永远小于等于它的扩展节点n'的估价函数值时，则启发式函数设计是单调的。

$$h(n) \le \cos t(n, n') + h(n')$$

启发式函数设计

- 不同的应用场景下有很多可选择的启发式函数。比如在网格地图中，一般使用以下几种启发式函数：
- 举个例子：

在正方形网格中，允许向4邻域的移动，使用曼哈顿距离（L1）

在正方形网格中，允许向8邻域的移动，使用对角线距离（L∞）

等等

启发函数没有限制，大家可以多尝试几种。

启发式函数设计

- 曼哈顿距离

```
function heuristic(node) =
    // dx
 = abs(node.x - goal.x)
    // dy
 = abs(node.y - goal.y)

     //return
 D * (dx + dy)
```

启发函数可以是曼哈顿距离的D倍。D的设计是为了距离衡量单位与启发函数相匹配。可以设置为方格间移动的最低代价值。

启发式函数设计

- 对角线距离（切比雪夫距离）

```
function heuristic(node) =

    dx
 = abs(node.x - goal.x)

    dy
 = abs(node.y - goal.y)

    return
 D * max(dx, dy)
```

如果在地图中允许朝着对角线方向移动，则可以考虑使用对角线距离。

内容

- 理论课内容回顾
  - 启发式搜索
  - A*
  - IDA*
  - 启发式函数设计
- 实验课任务与报告提交

实验任务

- 使用A*与IDA*算法解决15-Puzzle问题，启发式函数可以自己选取，最好多尝试几种不同的启发式函数
- 代码要求使用python 或者C++
- 报告要求

① 报告中需要包含对两个算法的原理解释

② 需要包含性能和结果的对比和分析

③ 如果使用了多种启发式函数，最好进行对比和分析

④ 需要在报告中分情况分析估价值和真实值的差距会造成算法性能差距的原因。（参考PPT P10）

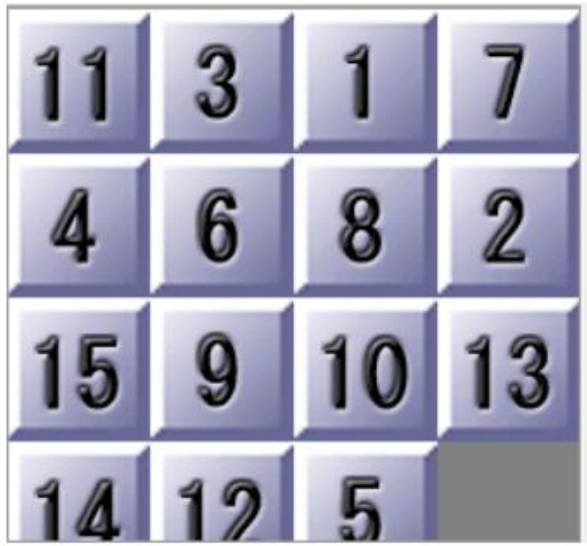
- 提交文件

① 实验报告：16******_wangxiaoming.pdf

② 代码：16******_wangxiaoming.zip 如果代码分成多个文件，需要写readme

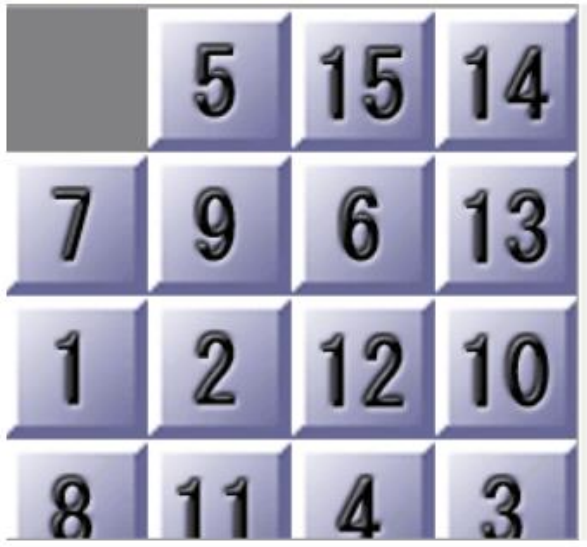
- DDL：2018-11-13 23:00:00

实验任务



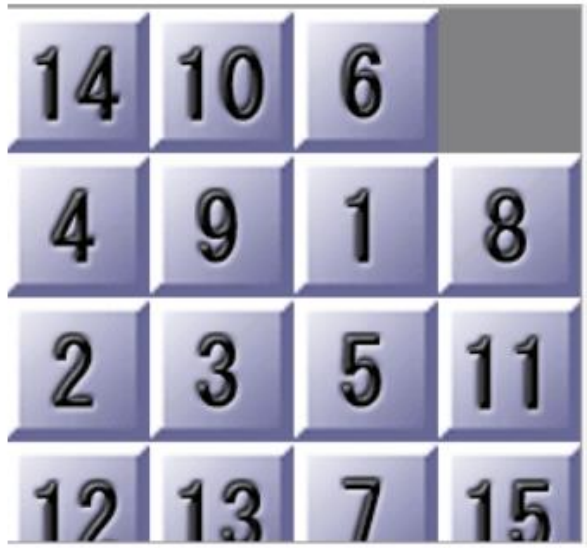
TextOut of Result

```
 11  3  1  7
  4  6  8  2
 15  9 10 13
 14 12  5  0
LowerBound 36 moves
A optimal solution 56 moves
Used time 3 sec
 13 10  8  6  9 12  5 13 10  8
 12 15 14  5 13 12 15 14  5 13
 14  9  4 11  3  1  6  4 11  3
  1  6  4  2  8 10 12 15 10  8
  7  4  2 11  3  5  9 10 11  3
  6  2  3  7  8 12
```



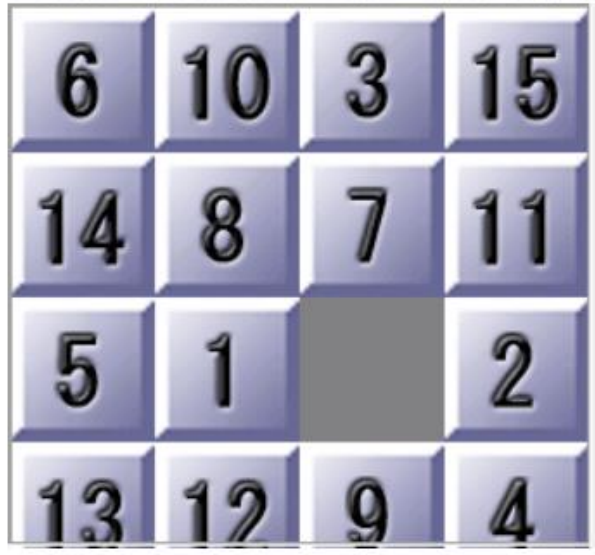
TextOut of Result

```
  0  5 15 14
  7  9  6 13
  1  2 12 10
  8 11  4  3
LowerBound 44 moves
A optimal solution 62 moves
Used time 4 sec
  7  9  2  1  9  2  5  7  2  5
  1 11  8  9  5  1  6 12 10  3
  4  8 11 10 12 13  3  4  8 12
 13 15 14  3  4  8 12 13 15 14
  7  2  1  5 10 11 13 15 14  7
  3  4  8 12 15 14 11 10  9 13
 14 15
```



TextOut of Result

```
 14 10  6  0
  4  9  1  8
  2  3  5 11
 12 13  7 15
LowerBound 37 moves
A optimal solution 49 moves
Used time 0 sec
  6 10  9  4 14  9  4  1 10  4
  1  3  2 14  9  1  3  2  5 11
  8  6  4  3  2  5 13 12 14 13
 12  7 11 12  7 14 13  9  5 10
  6  8 12  7 10  6  7 11 15
```



TextOut of Result

```
  6 10  3 15
 14  8  7 11
  5  1  0  2
 13 12  9  4
LowerBound 32 moves
A optimal solution 48 moves
Used time 0 sec
  9 12 13  5  1  9  7 11  2  4
 12 13  9  7 11  2 15  3  2 15
  4 11 15  8 14  1  5  9 13 15
  7 14 10  6  1  5  9 13 14 10
  6  2  3  4  8  7 11 12
```

Thanks!

Wanjun Zhong Nov.7