python实现的基于蒙特卡洛树搜索(MCTS)与UCB的五子棋游戏

转载 2017年02月22日 23:38:44 557

转载自http://www.cnblogs.com/xmwd/archive/2017/02/19/python_game_based_on_MCTS_and_UCB.html

MCTS与UCT

下面的内容引用自徐心和与徐长明的论文《计算机博弈原理与方法学概述》：

> 蒙特卡洛模拟对局就是从某一棋局出发，随机走棋。有人形象地比喻，让两个傻子下棋，他们只懂得棋规，不懂得策略，最终总是可以决出胜负。这个胜负是有偶然性的。但是如果让成千上万对傻子下这盘棋，那么结果的统计还是可以给出该棋局的固有胜率和胜率最高的着法。
> 蒙特卡洛树搜索通过迭代来一步步地扩展博弈树的规模，UCT 树是不对称生长的，其生长顺序也是不能预知的。它是根据子节点的性能指标导引扩展的方向，这一性能指标便是 UCB 值。它表示在搜索过程中既要充分利用已有的知识，给胜率高的节点更多的机会，又要考虑探索那些暂时胜率不高的兄弟节点，这种对于“利用”（Exploitation）和“探索”（Exploration）进行权衡的关系便体现在 UCT 着法选择函数的定义上，即子节点N_{i} 的 UCB 值按如下公式计算：

$$\frac{W_i}{N_i}+\sqrt{\frac{C\times lnN}{N_i}}$$

其中：

W_i：子节点获胜的次数；

N_i：子节点参与模拟的次数；

N：当前节点参与模拟的次数

C：加权系数。

可见 UCB 公式由两部分组成，其中前一部分就是对已有知识的利用，而后一部分则是对未充分模拟节点的探索。`C` 小偏重利用；而 `C` 大则重视探索。需要通过实验设定参数来控制访问节点的次数和扩展节点的阈值。

后面可以看到，在实际编写代码时，当前节点指的并不是具体的着法，而是当前整个棋局，其子节点才是具体的着法，它势必参与了每个子节点所参与的模拟，所以 `N` 就等于其所有子节点参与模拟的次数之和。当 `C` 取1.96时，置信区间的置信度达到95%，也是实际选择的值。

蒙特卡洛树搜索（MCTS）仅展开根据 UCB 公式所计算过的节点，并且会采用一种自动的方式对性能指标好的节点进行更多的搜索。具体步骤概括如下：
1.由当前局面建立根节点，生成根节点的全部子节点，分别进行模拟对局；
2.从根节点开始，进行最佳优先搜索；
3.利用 UCB 公式计算每个子节点的 UCB 值，选择最大值的子节点；
4.若此节点不是叶节点，则以此节点作为根节点，重复 2；
5.直到遇到叶节点，如果叶节点未曾经被模拟对局过，对这个叶节点模拟对局；否则为这个叶节点随机生成子节点，并进行模拟对局；
6.将模拟对局的收益（一般胜为 1 负为 0）按对应颜色更新该节点及各级祖先节点，同时增加该节点以上所有节点的访问次数；
7.回到 2，除非此轮搜索时间结束或者达到预设循环次数；
8.从当前局面的子节点中挑选平均收益最高的给出最佳着法。
由此可见 UCT 算法就是在设定的时间内不断完成从根节点按照 UCB 的指引最终走到某一个叶节点的过程。而算法的基本流程包括了选择好的分支（Selection）、在叶子节点上扩展一层（Expansion）、模拟对局（Simulation）和结果回馈（Backpropagation）这样四个部分。
UCT 树搜索还有一个显著优点就是可以随时结束搜索并返回结果，在每一时刻，对 UCT 树来说都有一个相对最优的结果。

代码实现

oard 类

`Board` 类用于存储当前棋盘的状态，它实际上也是MCTS算法的根节点。

```
class Board(object):
    """
    board for game
    """

    def __init__(self, width=8, height=8, n_in_row=5):
        self.width = width
        self.height = height
        self.states = {} # 记录当前棋盘的状态，键是位置，值是棋子，这里用玩家来表示棋子类型
        self.n_in_row = n_in_row # 表示几个相同的棋子连成一线算作胜利

    def init_board(self):
        if self.width < self.n_in_row or self.height < self.n_in_row:
            raise Exception('board width and height can not less than %d' % self.n_in_row) # 棋盘不能过小


        self.availables = list(range(self.width * self.height)) # 表示棋盘上所有合法的位置，这里简单的认为空的位置即合法

        for m in self.availables:
            self.states[m] = -1 # -1表示当前位置为空

    def move_to_location(self, move):
        h = move  // self.width
        w = move  %  self.width
        return [h, w]

    def location_to_move(self, location):
        if(len(location) != 2):
            return -1
        h = location[0]
        w = location[1]
        move = h * self.width + w
        if(move not in range(self.width * self.height)):
            return -1
        return move

    def update(self, player, move): # player在move处落子，更新棋盘
        self.states[move] = player
        self.availables.remove(move)
```

MCTS 类

核心类，用于实现基于UCB的MCTS算法。

```
class MCTS(object):
    """
    AI player, use Monte Carlo Tree Search with UCB
    """

    def __init__(self, board, play_turn, n_in_row=5, time=5, max_actions=1000):

        self.board = board
        self.play_turn = play_turn # 出手顺序
        self.calculation_time = float(time) # 最大运算时间
        self.max_actions = max_actions # 每次模拟对局最多进行的步数
        self.n_in_row = n_in_row

        self.player = play_turn[0] # 轮到电脑出手，所以出手顺序中第一个总是电脑
        self.confident = 1.96 # UCB中的常数
        self.plays = {} # 记录着法参与模拟的次数，键形如(player, move)，即（玩家，落子）
        self.wins = {} # 记录着法获胜的次数
        self.max_depth = 1

    def get_action(self): # return move

        if len(self.board.availables) == 1:
            return self.board.availables[0] # 棋盘只剩最后一个落子位置，直接返回

        # 每次计算下一步时都要清空plays和wins表，因为经过AI和玩家的2步棋之后，整个棋盘的局面发生了变化，原来的记录已经不适用了—原先普通的一步现在可能是致胜的一步，如果不清空，会影响现在的结果，导致这一步可能没那么“致胜”了
        self.plays = {}
        self.wins = {}
        simulations = 0
        begin = time.time()
        while time.time() - begin < self.calculation_time:
            board_copy = copy.deepcopy(self.board)  # 模拟会修改board的参数，所以必须进行深拷贝，与原board进行隔离
            play_turn_copy = copy.deepcopy(self.play_turn) # 每次模拟都必须按照固定的顺序进行，所以进行深拷贝防止顺序被修改
            self.run_simulation(board_copy, play_turn_copy) # 进行MCTS
            simulations += 1

        print("total simulations=", simulations)

        move = self.select_one_move() # 选择最佳着法
        location = self.board.move_to_location(move)
        print('Maximum depth searched:', self.max_depth)

        print("AI move: %d,%d\n" % (location[0], location[1]))

        return move

    def run_simulation(self, board, play_turn):
        """
        MCTS main process
        """

        plays = self.plays
        wins = self.wins
        availables = board.availables

        player = self.get_player(play_turn) # 获取当前出手的玩家
        visited_states = set() # 记录当前路径上的全部着法
        winner = -1
        expand = True

        # Simulation
        for t in range(1, self.max_actions + 1):
            # Selection
            # 如果所有着法都有统计信息，则获取UCB最大的着法
```

```
            if all(plays.get((player, move)) for move in availables):
                log_total = log(
                    sum(plays[(player, move)] for move in availables))
                value, move = max(
                    ((wins[(player, move)] / plays[(player, move)]) +
                     sqrt(self.confident * log_total / plays[(player, move)]), move)
                    for move in availables)
            else:
                # 否则随机选择一个着法
                move = choice(availables)

            board.update(player, move)

            # Expand
            # 每次模拟最多扩展一次，每次扩展只增加一个着法
            if expand and (player, move) not in plays:
                expand = False
                plays[(player, move)] = 0
                wins[(player, move)] = 0
                if t > self.max_depth:
                    self.max_depth = t

            visited_states.add((player, move))

            is_full = not len(availables)
            win, winner = self.has_a_winner(board)
            if is_full or win: # 游戏结束，没有落子位置或有玩家获胜
                break

            player = self.get_player(play_turn)

        # Back-propagation
        for player, move in visited_states:
            if (player, move) not in plays:
                continue
            plays[(player, move)] += 1 # 当前路径上所有着法的模拟次数加1
            if player == winner:
                wins[(player, move)] += 1 # 获胜玩家的所有着法的胜利次数加1

    def get_player(self, players):
        p = players.pop(0)
        players.append(p)
        return p

    def select_one_move(self):
        percent_wins, move = max(
            (self.wins.get((self.player, move), 0) /
             self.plays.get((self.player, move), 1),
             move)
            for move in self.board.availables) # 选择胜率最高的着法

        return move

    def has_a_winner(self, board):
        """
        检查是否有玩家获胜
        """
        moved = list(set(range(board.width * board.height)) - set(board.availables))
        if(len(moved) < self.n_in_row + 2):
            return False, -1

        width = board.width
        height = board.height
        states = board.states
        n = self.n_in_row
        for m in moved:
```

```
            w = m % width
            player = states[m]

            if (w in range(width - n + 1) and
                len(set(states[i] for i in range(m, m + n))) == 1): # 横向连成一线
                return True, player

            if (h in range(height - n + 1) and
                len(set(states[i] for i in range(m, m + n * width, width))) == 1): # 竖向连成一线
                return True, player

            if (w in range(width - n + 1) and h in range(height - n + 1) and
                len(set(states[i] for i in range(m, m + n * (width + 1), width + 1))) == 1): # 右斜向上
连成一线
                return True, player

            if (w in range(n - 1, width) and h in range(height - n + 1) and
                len(set(states[i] for i in range(m, m + n * (width - 1), width - 1))) == 1): # 左斜向下
连成一线
                return True, player

        return False, -1

    def __str__(self):
        return "AI"
```

Human 类

用于获取玩家的输入，作为落子位置。

```
class Human(object):
    """
    human player
    """

    def __init__(self, board, player):
        self.board = board
        self.player = player

    def get_action(self):
        try:
            location = [int(n, 10) for n in input("Your move: ").split(",")]
            move = self.board.location_to_move(location)
        except Exception as e:
            move = -1
        if move == -1 or move not in self.board.availables:
            print("invalid move")
            move = self.get_action()
        return move

    def __str__(self):
        return "Human"
```

Game 类

控制游戏的进行，并在终端显示游戏的实时状态。

```
class Game(object):
    """
    game server
    """

    def __init__(self, board, **kwargs):
        self.board = board
        self.player = [1, 2] # player1 and player2
        self.n_in_row = int(kwargs.get('n_in_row', 5))
        self.time = float(kwargs.get('time', 5))
        self.max_actions = int(kwargs.get('max_actions', 1000))

    def start(self):
        p1, p2 = self.init_player()
        self.board.init_board()

        ai = MCTS(self.board, [p1, p2], self.n_in_row, self.time, self.max_actions)
        human = Human(self.board, p2)
        players = {}
        players[p1] = ai
        players[p2] = human
        turn = [p1, p2]
        shuffle(turn) # 玩家和电脑的出手顺序随机
        while(1):
            p = turn.pop(0)
            turn.append(p)
            player_in_turn = players[p]
            move = player_in_turn.get_action()
            self.board.update(p, move)
            self.graphic(self.board, human, ai)
            end, winner = self.game_end(ai)
            if end:
                if winner != -1:
                    print("Game end. Winner is", players[winner])
                break

    def init_player(self):
        plist = list(range(len(self.player)))
        index1 = choice(plist)
        plist.remove(index1)
        index2 = choice(plist)

        return self.player[index1], self.player[index2]

    def game_end(self, ai):
        """
        检查游戏是否结束
        """
        win, winner = ai.has_a_winner(self.board)
        if win:
            return True, winner
        elif not len(self.board.availables):
            print("Game end. Tie")
            return True, -1
        return False, -1

    def graphic(self, board, human, ai):
        """
        在终端绘制棋盘，显示棋局的状态
        """
        width = board.width
        height = board.height

        print("Human Player", human.player, "with X".rjust(3))
        print("AI    Player", ai.player, "with O".rjust(3))
        print()
        for x in range(width):
```

```
            print("{0:8}".format(x), end='')
        print('\r\n')
        for i in range(height - 1, -1, -1):
            print("{0:4d}".format(i), end='')
            for j in range(width):
                loc = i * width + j
                if board.states[loc] == human.player:
                    print('X'.center(8), end='')
                elif board.states[loc] == ai.player:
                    print('O'.center(8), end='')
                else:
                    print('_'.center(8), end='')
            print('\r\n\r\n')
```

增加简单策略

实际运行时，当棋盘较小（6X6），需要连成一线的棋子数量较少（4）时，算法发挥的水平不错，但是当棋盘达到8X8进行五子棋游戏时，即使将算法运行的时间调整到10秒，算法的发挥也不太好，虽然更长的时间效果会更好，但是游戏体验就实在是差了。因此考虑增加一个简单的策略：当不是所有着法都有统计信息时，不再进行随机选择，而是优先选择那些在当前棋盘上已有落子的邻近位置中没有统计信息的位置进行落子，然后选择那些离得远的、没有统计信息的位置进行落子，总得来说就是尽可能快速地让所有着法具有统计信息。对于五子棋来说，关键的落子位置不会离现有棋子太远。

下面是引入新策略的代码：

```
def run_simulation(self, board, play_turn):

    for t in range(1, self.max_actions + 1):
        if ...
            ...
        else:
            adjacents = []
            if len(availables) > self.n_in_row:
                adjacents = self.adjacent_moves(board, player, plays) # 没有统计信息的邻近位置

            if len(adjacents):
                move = choice(adjacents)
            else:
                peripherals = []
                for move in availables:
                    if not plays.get((player, move)):
                        peripherals.append(move) # 没有统计信息的外围位置
                move = choice(peripherals)
    ...

def adjacent_moves(self, board, player, plays):
    """
    获取当前棋局中所有棋子的邻近位置中没有统计信息的位置
    """
    moved = list(set(range(board.width * board.height)) - set(board.availables))
    adjacents = set()
    width = board.width
    height = board.height

    for m in moved:
        h = m // width
        w = m % width
        if w < width - 1:
            adjacents.add(m + 1) # 右
        if w > 0:
            adjacents.add(m - 1) # 左
        if h < height - 1:
            adjacents.add(m + width) # 上
        if h > 0:
            adjacents.add(m - width) # 下
        if w < width - 1 and h < height - 1:
            adjacents.add(m + width + 1) # 右上
        if w > 0 and h < height - 1:
            adjacents.add(m + width - 1) # 左上
        if w < width - 1 and h > 0:
            adjacents.add(m - width + 1) # 右下
        if w > 0 and h > 0:
            adjacents.add(m - width - 1) # 左下

    adjacents = list(set(adjacents) - set(moved))
    for move in adjacents:
        if plays.get((player, move)):
            adjacents.remove(move)
    return adjacents
```

现在算法的效果就有所提升了。

蒙特卡洛树搜索 **MCTS (http://blog.csdn.net/zkl99999/article/details/50677996)**

字数2205 阅读46 评论0 喜欢1 源地址0. http://www.jianshu.com/users/696dc6c6f01c/latest_articles 什么是 MCTS？ ...

zkl99999 (http://blog.csdn.net/zkl99999) 2016年02月17日 01:21 4192

蒙特卡洛树算法 **(MCTS) (http://blog.csdn.net/Jaster_wisdom/article/details/50845090)**

实质上可以看成一种增强学习蒙特卡罗树搜索(MCTS)会逐渐的建立一颗不对称的树。可以分为四步并反复迭代： (1)选择从根节点，也就是要做决策的局面R出发向下选择一个最急迫需要...

Jaster_wisdom (http://blog.csdn.net/Jaster_wisdom) 2016年03月10日 11:29 7840

那么蒙特卡洛树搜索**(Monte Calro Tree Search, MCTS)**究竟是啥 **(http://blog.csdn.net/na…**

同时发布于：http://www.longgaming.com/archives/214Intro最近阿法狗和李师师的人机大战着实火了一把，还顺带捧红了柯杰，古力等一干九段。虽然我从小学的是象棋，对围...

natsu1211 (http://blog.csdn.net/natsu1211) 2016年03月26日 14:41 11953

蒙特卡洛树搜索介绍 **(http://blog.csdn.net/z84616995z/article/details/50915952)**

2015年9月7日周一 由Jeff Bradberry留 与游戏AI有关的问题一般开始于被称作完全信息博弈的游戏。这是一款对弈玩家彼此没有信息可以隐藏的回合制游戏且在游戏技术里没有运气元...

z84616995z (http://blog.csdn.net/z84616995z) 2016年04月03日 12:41 4260

蒙特卡洛树搜索 **MCTS (http://blog.csdn.net/AMDS123/article/details/71038316)**

什么是 MCTS？全称 Monte Carlo Tree Search，是一种人工智能问题中做出最优决策的方法，一般是在组合博弈中的行动（move）规划形式。它结合了随机模拟的一般性和树搜索的准确...

AMDS123 (http://blog.csdn.net/AMDS123) 2017年05月01日 02:46 5998

UCT（信心上限树算法）解四子棋问题——蒙特卡罗法模拟人机博弈 **(http://blog.csdn.…**

虽说UC

u014397729 (http://blog.csdn.net/u014397729) 2014年05月28日 23:10 4642

十四步实现强大的五子棋**AI (http://blog.csdn.net/qwerty_xk/article/details/8498705)**

十四步实现拥有强大AI的五子棋游戏又是本人一份人工智能作业……首先道歉，从Word贴到Livewrter，好多格式没了，也没做代码高亮……大家凑活着看……想做个好的人机对弈的五子棋，可以说...

qwerty_xk (http://blog.csdn.net/qwerty_xk) 2013年01月13日 19:48 2685

Python金融大数据分析**-**蒙特卡洛仿真 **(http://blog.csdn.net/qtlyx/article/details/53613315)**

1.简单的例子了解一点金融工程的对这个公式都不会太陌生，是用现在股价预测T时间股价的公式，其背后是股价符合几何布朗运动，也就是大名鼎鼎的BSM期权定价模型的基础。我们假设现在一个股票的价值是1...

qtlyx (http://blog.csdn.net/qtlyx) 2016年12月14日 17:06 3855

蒙特卡洛树搜索**(MCTS)**进行模拟的实现流程 **(http://blog.csdn.net/u010296599/article/de…**

首先，要明确的一点是，算法并不用了解游戏的领域知识。 在一个游戏模拟过程中，相关决策的组合可能是一个很大的数，我们如何控制这个模拟行为是满足一定时间上的限制的。我们允许一个参数来控制时间...

u010296599 (http://blog.csdn.net/u010296599) 2017年03月01日 16:58 568

模仿**AlphaGo**围棋博弈，**MuGo**实现策略网络以及蒙特卡洛树搜索 **(http://blog.csdn.net/…**

AlphaGo数次击败了人类围棋高手，大家是不是摩拳擦掌，想对Alphago一探究竟。网友brliee的作品MuGO实现了AlphaGo的策略网络(policy)和蒙塔卡罗树搜索(mcts)两大主...

qizongshuai (http://blog.csdn.net/qizongshuai) 2017年08月10日 22:54 789

增强学习**Reinforcement Learning**经典算法梳理**2**：蒙特卡洛方法 **(http://blog.csdn.net/s…**

1 前言在上一篇文章中，我们介绍了基于Bellman方程而得到的Policy Iteration和Value Iteration两种基本的算法，但是这两种算法实际上很难直接应用，原因在于依然是偏于理想...

songrotek (http://blog.csdn.net/songrotek) 2016年05月12日 10:17 11149

使用 **Python** 搭建简易版**AlphaGo (http://blog.csdn.net/willduan1/article/details/54311778)**

参考文献： http://mp.weixin.qq.com/s?__biz=MzA3MzI4MjgzMw==&mid=2650722126&idx=4&sn=96b2fa1fe3fc858d415dd...

willduan1 (http://blog.csdn.net/willduan1) 2017年01月13日 13:31 9468

围棋**AI**之路（三）**UCT**，进来之后才发现是地狱 **(http://blog.csdn.net/oyd/article/details…**

照例还是先公布代码 http://download.csdn.net/source/913373以及编译好的可执行程序，下载地址：http://download.csdn.net/source/913...

oyd (http://blog.csdn.net/oyd) 2008年12月27日 00:11 10106

AlphaGo背后的搜索算法：蒙特卡罗树搜索 **&& alphago** 代码 **(http://blog.csdn.net/Real…**

代码： https://github.com/Rochester-NRT/AlphaGo AlphaGo背后的搜索算法：蒙特卡罗树搜索 ...

Real_Myth (http://blog.csdn.net/Real_Myth) 2016年03月14日 11:07 3553

bandit算法原理及Python实现 (http://blog.csdn.net/z1185196212/article/details/53374194)

bandit算法、多臂老虎机问题、在线学习

z1185196212 (http://blog.csdn.net/z1185196212) 2016年11月28日 09:53 3645

pythonAI五子棋 (http://blog.csdn.net/weixin_37720172/article/details/78236154)

pythonAI五子棋大概一个半月前，突发奇想写一个AI五子棋，顺便熟悉下机器学习的内容。经过一个多月的努力（其实我已没有天天在写，有点想法了就写写），我终于放弃了。是的，我放弃了！！！坚持就是胜利，...

weixin_37720172 (http://blog.csdn.net/weixin_37720172) 2017年10月14日 19:12 785

400行代码实现双人对战五子棋(适合新手入门) (http://blog.csdn.net/qq_26971803/articl…

400行代码实现双人对战五子棋(适合新手入门)跟上一篇博客一样,都是看了慕课网的视频之后写的学习记录,记录一下实现的思路,大部分内容比较简单,但也从中学到了很多东西.从中能学到的知识点: Androi...

qq_26971803 (http://blog.csdn.net/qq_26971803) 2016年04月05日 00:40 5574

python实现五子棋游戏程序 (http://download.csdn.net/download/weixi…

(http://download.csdn.net/ 2017年03月21日 23:04 96KB 下载 (

百行内实现五子棋人机对战 **(http://blog.csdn.net/skywind/article/details/8164713)**

国庆没事，想看看最少多少行可以写一个人机对战起来游戏，于是有了这个Python版五子棋人机对战，仅仅几百行。 #! /usr/bin/env python # -*- coding: u...

skywind (http://blog.csdn.net/skywind) 2012年11月09日 03:51 11591

python大作业 五子棋 ai (http://download.csdn.net/download/w1135181…

(http://download.csdn.net/ 2013年12月17日 19:58 15KB 下载 (