HSEA-hw2-实验报告

丁豪 (181220010、181220010@smail.nju.edu.cn)

(南京大学 人工智能学院, 南京 210093)

摘 要: 本次实验基于 RHEA 与 Self-Adaptive RHEA 两篇论文，理解、实现了不同参数环境下使用 macro action 代替普通 action 的演化算法以及自适应调参,并且与其他传统解决此类问题的通用方法进行了性能对比. 最终我们发现使用演化算法的 Agent 无论在速度、分数还是稳定性上均有较好的表现。

关键词: 演化算法；自动调参；gvgai

中图法分类号: TP301 **文献标识码:** A

1 任务一

1.1 伪代码

```
class Algorithm():
    # 1.rolling-horizon获取动作框架
    def GetAction(self, gs):
        if self.IsGameFirstAction(gs):
            self.actionToRun = self.DecideMacroAction(gs)
        else:
            GS = gs.copy()
            for i in range(self.remainingActions):
                GS.Advance(self.actionToRun)
            if self.remainingActions > 0:
                if self.resetAlgorighm:
                    self.reset(GS)
                    self.resetAlgorighm = False
                self.nextMove(GS)
            else:
                self.actionToRun = self.DecideMacroAction()
        self.remainingActions -= 1
        return self.actionToRun
    def DecideMacroAction(self,gs):
        self.actionToRun = self.nextMove(gs)
        self.remainingActions = self.L
        self.resetAlgorighm = True
        return self.actionToRun
```

```
# 2.演化算法的实现
def nextMove(self, gs):
    population = randomly generate 10 integer vector of size self.N
    while time is enough do:
        evaluate each individual in population by self.VALUE(gs,individual)
        next_population = []
        next_population.add(best 2 of population by VALUE)
        while size(new_population < 10) do:
            parent1, parent2 = tournament(size = 3, population)
            children = uniform_crossover(parent1,parent2)
            children = mutate(children)
            next_population.add(children)
        population = next_population
    return firstmove(best(population))
def Value(self, gs, individual):
    for macro_action in individual:
        for i in range(L):
            GS = gs.advance(macro_action)
    v1 = 10000 - GS.dw
    v2 = 1000 * GS.waypoints_visited
    v3 = 10000 - GS.timespent
    v4 = -100 * GS.collisions

    return v1+v2+v3+v4
```

1.2 描述演化算法部件

1.2.1 解的表示

在该篇论文中，行动空间是离散化的 6 个动作（分为两维输入，加速度 on off，方向 left right straight，总共 2x3=6 种组合动作），同时为了性能需要，连续的 L 个动作被抽象成一个“宏动作”。解的表示空间就是建立在宏动作空间上的。每一个解包含 N 个宏动作，因此使用整数表示，是一个 N 维整数向量。这其中涉及到两个常亮 L 与 N，均是在实验阶段设定的超参数。

1.2.2 种群

初始为 10 个随机基因。后续维护大小为 10 的种群。

1.2.3 交叉算子

Uniform Crossover

1.2.4 变异算子

对 0-5 中的每一个组合动作，一次只选择其“加速”或“方向”分量进行变异。如果选择对“加速”进行变异，则以一定概率反转加速属性，即“on”与“off”互相变化。如果选择对“方向”进行变异，“向左、向右”变异为“向前”，而向前则分别以一半的概率变异为“向左”或“向右”。

1.2.5 解的评估

引入了 Score Function 来进行评估，评估对象是执行完相应连续操作之后的状态。此函数包含四个分量，分别为离 waypoint 的距离评分、已经到达的 waypoint 数量评分、消耗时间评分、碰撞次数评分，这些评分分别都有对应的权重超参数来控制其影响力。

1.2.6 解的选择

第一部分，在父带节点中使用 Elitism 直接推荐两个最佳个体进入下一代。第二部分，采用数量为 3 的 tournament selection 选择父代个体进行繁衍，然后从子代中挑选一定数量的最优个体，以补足种群数量。

2 任务二

2.1 简要阐述sampleRHEA的代码结构与算法流程

2.1.1 代码结构

Individual 类：实现了遗传算法中“个体”所需的元素和方法

- 元素：individual 的基因型（actions）序列，individual 的 value（由评估函数得到）
- crossover 方法：实现了 one_point 与 uniform 两种 crossover 方法。
- mutate 方法：选取随机点位，变异为随机合法动作，重复执行指定次数。
- compareTo 方法：重写了比较方法，使得两个 Individual 的比较按照其 value 进行。
- 其他方法：重置、输出、复制、判断相等

Agent 类：利用上述 individual，实现了利用演化算法进行决策的过程

- 元素：维护了“种群”及其相关属性，同时在初始化阶段指定了演化算法的诸多超参数。
- act 方法：在每次需要决策时调用，会返回当前种群所指示的最佳下一步动作。
- runIteration 方法：被 act 方法调用，执行一轮演化算法的迭代过程。
- evaluate 方法：使当前 state 按照 individual 指定的动作序列前进，并对最终到达的 state 根据启发式函数评价其分数。（这里的启发式函数使用的是简单的游戏输赢+游戏分数）
- crossover 方法：采用 tournament selection 方法选取父母，然后将其 crossover 得到子代。
- 其他方法：初始化，重置等。

2.1.2 算法流程

sampleRHEA 与 RHEA 论文中的算法流程基本相同，主要区别在于没有使用 macro Action，因而不需要对某个动作重复执行 L 次。这主要影响以下两个方面：

1：评估函数中只需依次执行 Individual 对应的 simulation_depth 个动作，即可得到需要评估的状态。

2：rolling-horizon 过程被改变，不再能够每 L 步时间进行一次决策，而要求每一步都要做决策。

此外，在变异的过程中，论文中是对子代按照一定概率变异，而 sampleRHEA 则是指定了变异数量。

在每一次决策过程中，agent 会在时间允许范围内不断调用 runIteration，来进行一轮演化算法迭代过程。迭代时父代的 elite 会直接保留，此外再重复执行用 tournament selection 挑选父母生成子代并变异，直到新种群数量达到所维护的定值。在时间即将结束时，Agent 从当前种群中挑选出最优个体，并将他的第一个动作返回。

2.2 挑选5个游戏

- 0 alien：射击飞船，动态消除类游戏。需要在最大化即时奖励的同时防止全局失败。
- 4 bait：推箱子，解谜游戏。走出迷宫往往需要长远的考虑，且动作粒度小，连续操作未必好。
- 69 pacoban：上次的作业，收集、生存游戏。存在 hack 方法（重复吃 ghost）能获得特殊高分。
- 97 towerdefence：塔防游戏。延迟奖励现象严重，且前期操作失误将有严重的延迟惩罚。
- 112 asteroids：二维平面打飞机游戏，连续状态、动作，无网格化。

2.3 实现MyRHEA

实现于 src/tracks/singlePlayer/myRHEA，包含 Agent、Individual、myHeuristic 三个类。

实现：主要是针对 sampleRHEA 添加了 macro Actions 的实现，更改了启发式迭代的条件使其可以在决定好某次 macro actions 后 remaining actions > 0 的情况下继续搜索。同时更新了 evaluate 函数模拟执行的过程，改为每个 action 重复执行 L 次。超参数 L 在初始化阶段指定。

改进：交叉算子改为 one_point_crossover，使得新生成的子代可以保留父母各自的局部信息（经过试验发现，往往连续的局部信息是有更用的）。同时修改了变异数量为 2，以增强子代的探索性。最后，父代选择时 tournament size 增大到 4，增加父代竞争性（更偏向于选择更好的父代）。

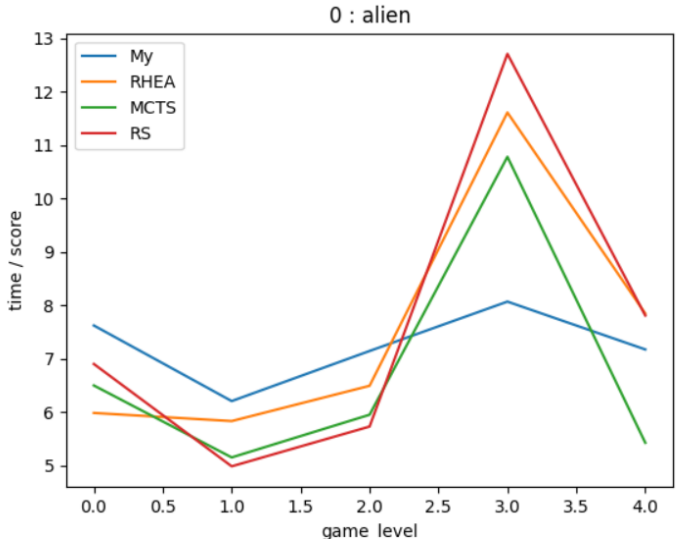
此外，更新了启发式函数，在其中加入了

1：Agent health point 指标，使得可以挑选不容易是自己死亡的行动。

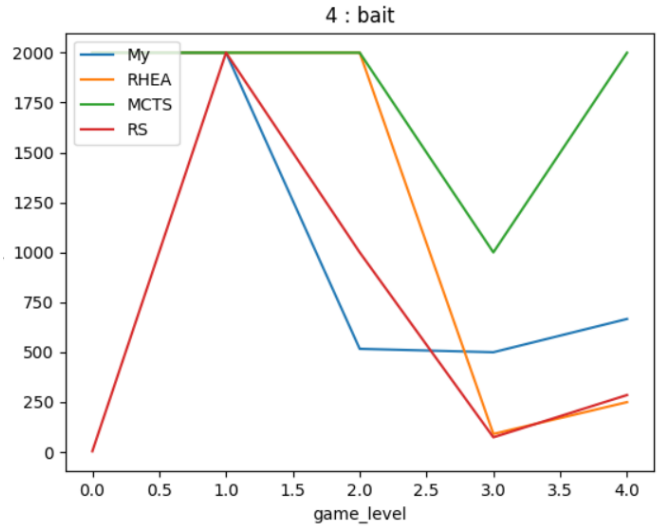
2：gametick 指标，使得 agent 倾向于执行耗时较少的操作。

2.4 性能对比测试

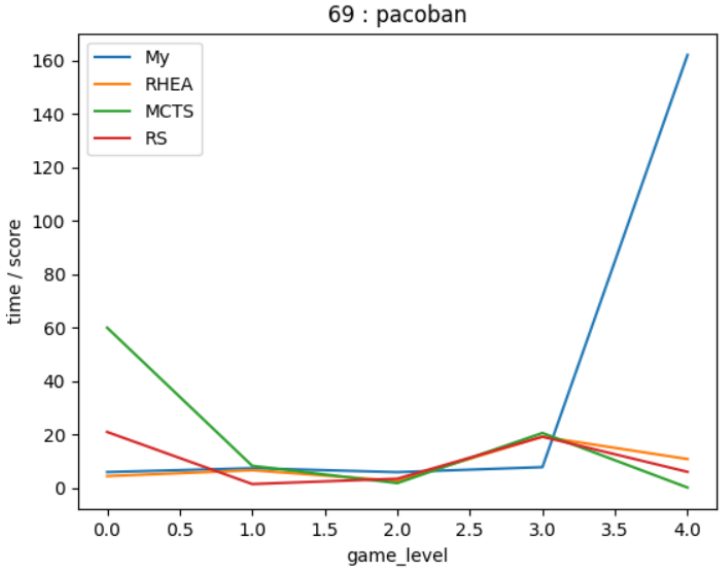
以下对 3.2 中所述的 5 个游戏，每个算法分别在不同 level 下运行 10 轮，取平均值进行绘图。图片的横轴是游戏 level，图片的纵轴为 average(game_tick / score)，用于反映**单位得分所需时间**。（为了防止除 0，所有得分均加一）此性能指标应当越低越好。



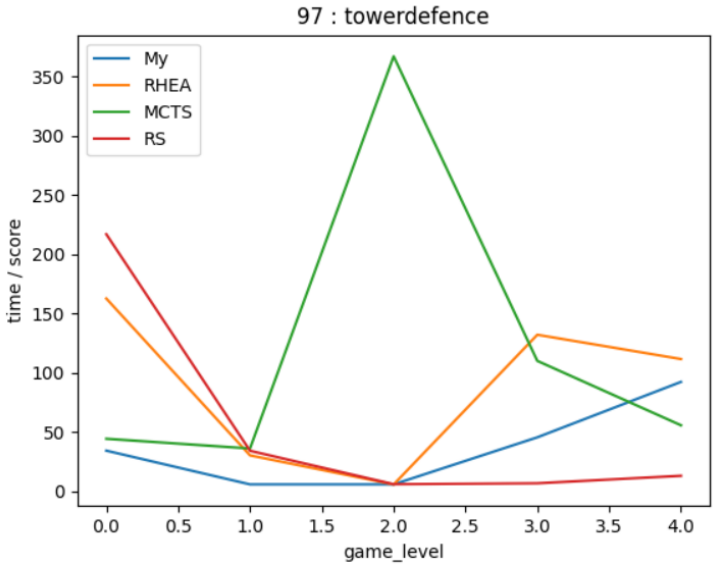
在 alien 游戏中，相比其余算法 myRHEA 不同 level 分数更为平均，但性能无明显优势。这可能是因为这个环境对 macro action 的适配性较为一般，有时可能会导致浪费时间。



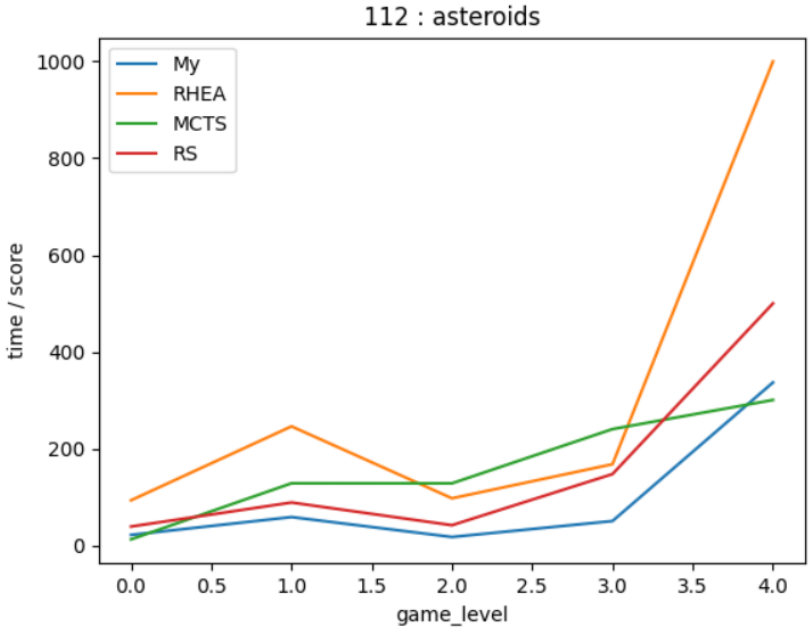
在 bait 游戏中，所有算法的表现均一般。特别关注到 myRHEA 算法经常陷入僵局，无法移动。这是因为连续执行 L 步相同操作，往往会让自己走入死路，或者直接结束。



在 pacoban 游戏中，几个算法表现较为一致，除了 myRHEA 在最高难度时，能吃到的金币尤其少。这里的主要原因与 bait 游戏相同，即使用了 macro action，限制了 avatar 可及位置的粒度，只能每 L 步到达一个位置，在最后一关中会直接把自己卡住无法移动。



在 towerdefence 中，除了 MCTS 在 level2 表现较差之外，其预算法性能大致相当。使用 Macro action 在这里加快了 agent 到达所需位置的速度，但是在进行“建造”命令的时候却会产生冗余操作，导致时间的浪费。



在 asteroids 游戏中，myRHEA 算法有着相当明显的优势。这主要是因为这个游戏环境与 RHEA 论文所描述的真实旅行商问题相当类似，是一个模拟的“连续行动、状态空间”环境。在这种环境下采取 macro action，帮助我们将连续行动的颗粒度提升，使每一次决策产生的影响性更大，进而更好的反映不同动作间的行动值差异。可以看出 myRHEA 算法对这类环境有着独特的优势。

3 任务三

3.1 Self-Adaptive RHEA的改进

基础的 RHEA 中，诸如 Genetic Operator, Selection Type, Crossover Type 等，都是预先指定好的超参数，在代码运行过程中是不变的。而 Self-Adaptive RHEA 则将这些参数使用 Tuner 在启发式迭代过程中不断根据其性能进行调整。

在每一轮迭代之前，RHEA Agent 从 Tunner 中得到超参数，进行一次迭代，然后 Tunner 根据新迭代产生的最佳子代与最佳父代适应性的差值来对超参数进行相应的调整。这样的改进好处有二，一是使得 Agent 具有了全局搜索最优超参数的能力而减轻初始超参数设置不妥产生的负面影响，二是能够不断跟进游戏局面的变化，选择更加适合当前环境的超参数，有点类似于在机器学习中使用动态学习率以实现更精细的调整。

3.2 Self-Adaptive RHEA的实现

实现于 src/tracks/singlePlayer/myAdaptiveRHEA，包含 Agent、Individual、myHeuristic、Tunner 四个类。Individual、myHeuristic 与原始 myRHEA 相同。Agent 中增加了从 tunner 获得超参数，以及在一轮迭代后用 value 差值让 tunner 更新超参数。Tunner 类实现了论文中的**使用 EA** 的超参数调整器。具体实现详见代码。

4 结束语

本次实验从论文出发，着眼于方法，落实于代码，帮助我们深入浅出的梳理了演化算法的全局优势与具体细节。本次实验美中不足是时间较为有限，未能进一步加强行之有效的启发式知识，如果能补足应当对最终性能有极大的提升。