

SWP - Data Science SS17

From Marlene Kreß, Janis Krzok
and Phillip Witte

Effective leadership and decision-making in animal groups on the move

Iain D. Couzin, Jens Krause, Nigel R. Franks & Simon A. Levin

Paper

- Movement decisions in groups of animals
- Informed agents (weight, preferred direction)
- Number of informed agents necessary to influence group
- Influence of different groups of informed agents

Model

- Based on Couzin model
- Zone of orientation and Zone of attraction combined

$$\mathbf{d}_i(t + \Delta t) = \sum_{j \neq i} \frac{\mathbf{c}_j(t) - \mathbf{c}_i(t)}{|\mathbf{c}_j(t) - \mathbf{c}_i(t)|} + \sum_{j=1} \frac{\mathbf{v}_j(t)}{|\mathbf{v}_j(t)|}$$

- For informed agents add preferred direction with weight

$$\mathbf{d}_i'(t + \Delta t) = \frac{\hat{\mathbf{d}}_i(t + \Delta t) + \omega \mathbf{g}_i}{|\hat{\mathbf{d}}_i(t + \Delta t) + \omega \mathbf{g}_i|}$$

- Weight feedback

Our Implementation

- Used classic couzin algorithm
- Pref direction:

```
#prefered direction  
absLen2 = absvec((dx + (a.weight * a.prefdir[0])) , (dy + (a.weight * a.prefdir[1])))  
dx = (dx + (a.weight * a.prefdir[0])) / absLen2  
dy = (dy + (a.weight * a.prefdir[1])) / absLen2
```

Our Implementation cont'

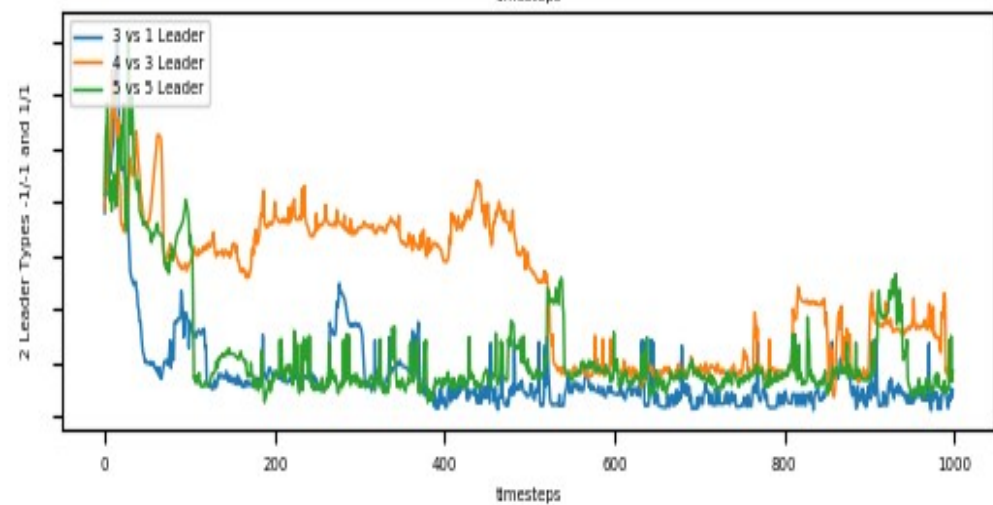
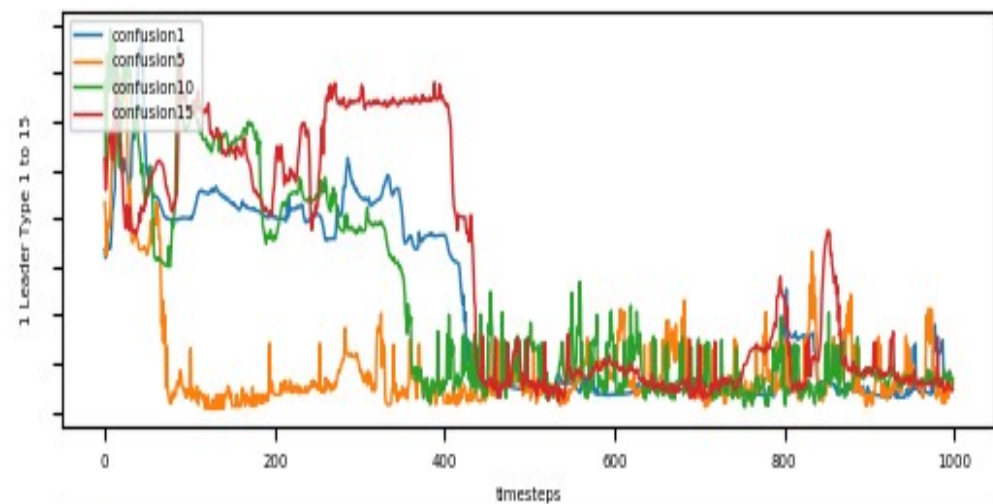
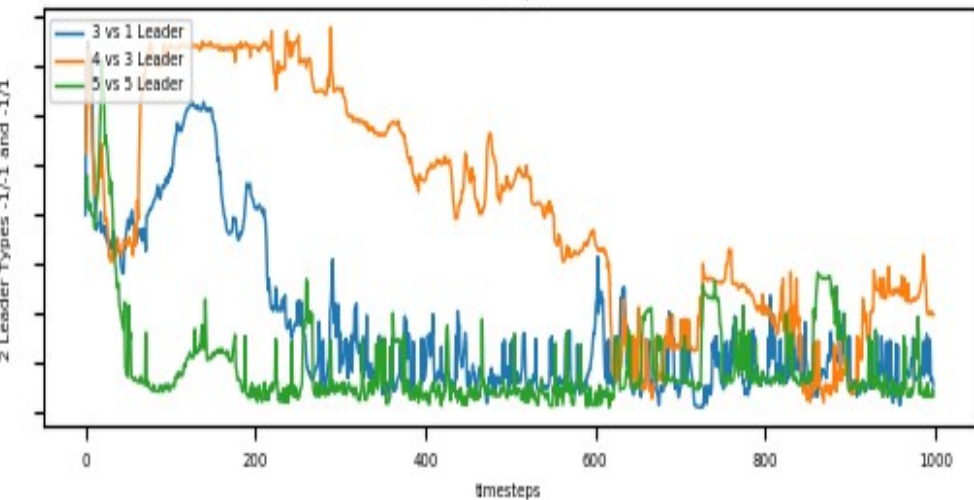
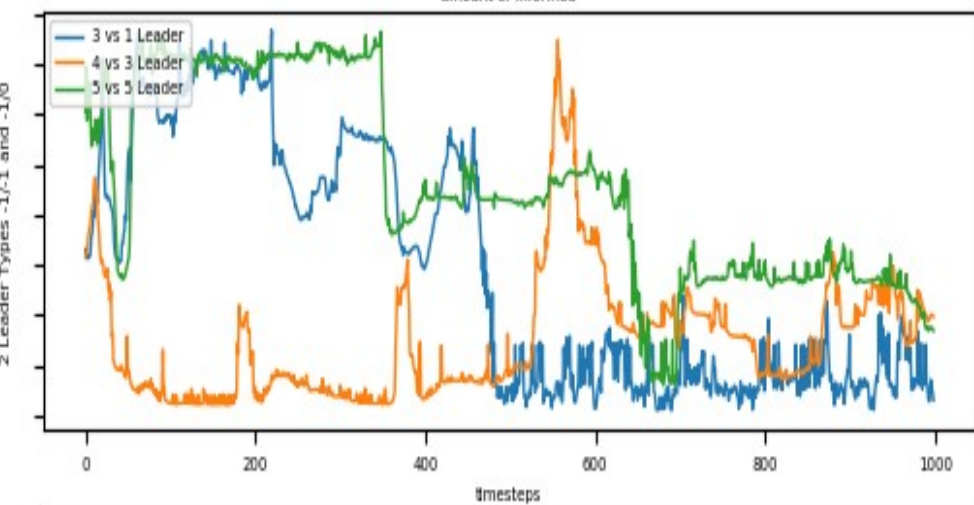
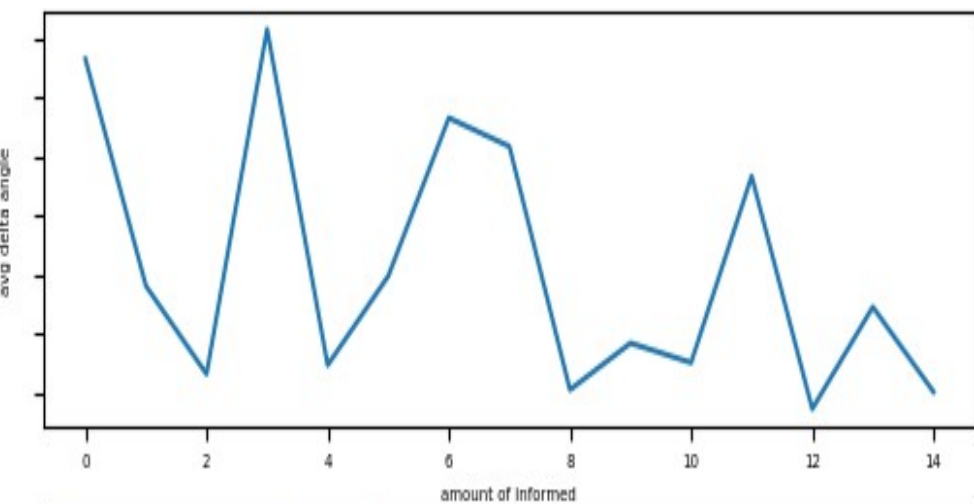
Weight feedback loop:

```
if (agent.prefdir != [0,0]):  
    angle_pref = math.atan2(agent.prefdir[0], agent.prefdir[1])  
    angle_new = math.atan2(agent.x_velocity, agent.y_velocity)  
    beta = math.degrees(angle_new - angle_pref)  
  
    if abs(beta) > 180:  
        if beta < 0:  
            beta += 360  
        else:  
            beta -= 360  
  
    if abs(beta) < 20 :  
        agent.set_weight(min(1, agent.weight+0.001))  
    else:  
        agent.set_weight(max(0, agent.weight-0.000001))
```

Quantification

Test cases

- for 1 group of informed agents
 - Group size 1 to 15
- For 2 groups of informed agents
 - Different directions, size proportion
3-1, 4-3, 5-5



Evolving the selfish herd:
emergence
of distinct aggregating strategies
in an individual-based model

Andrew J. Wood and Graeme J. Ackland

Paper

- Evolution of animal groups via genetic algorithm
- Influence of foraging and predation

Model

- Movement based on Couzin + food/predator bias
- First generation random
- Warm up time
- Introduce food / predator
- After each iteration: select agents, mutate, generate new generation

Model cont'

parameter	symbol	value or constraint	notes
system size	L	400	periodic BC
no. of boids	N	80	
time-step	Δt	0.1 s	
repulsion radius	R_r	1	fixed
orientation radius	R_o	$R_r < R_o < R_a$	evolvable
attraction radius	R_a	$R_a > \sqrt{A_s/2\pi}$	evolvable
speed	v	$1 < v_i < 5$	evolvable
viewing angle	θ	$\theta < 360^\circ$	$A_s/(R_a^i)^2$
turning angle	ϕ	$\phi < 180^\circ$	$A_m/2(v_i)^2$
food preference	Ω^f	free	evolvable
anti-predator preference	Ω^p	free	evolvable
noise	σ	free	evolvable

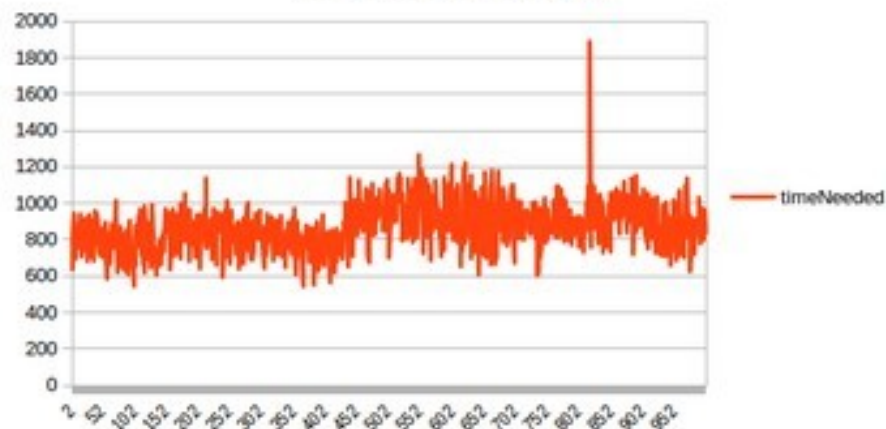
updates velocity, taking food and predators and other agents into account

```
def updateV_final(agent, matrix, foods, predators):  
    vc = updateV_couzin(agent, matrix)  
    vf = check_food(agent, foods)  
    vp = check_predator(agent, predators)  
  
    vvX = vc[0] + agent.food_pref* vf[0] - agent.anti_pred * vp[0]  
    vvY = vc[1] + agent.food_pref* vf[1] - agent.anti_pred * vp[1]  
  
    return [vvX, vvY]
```

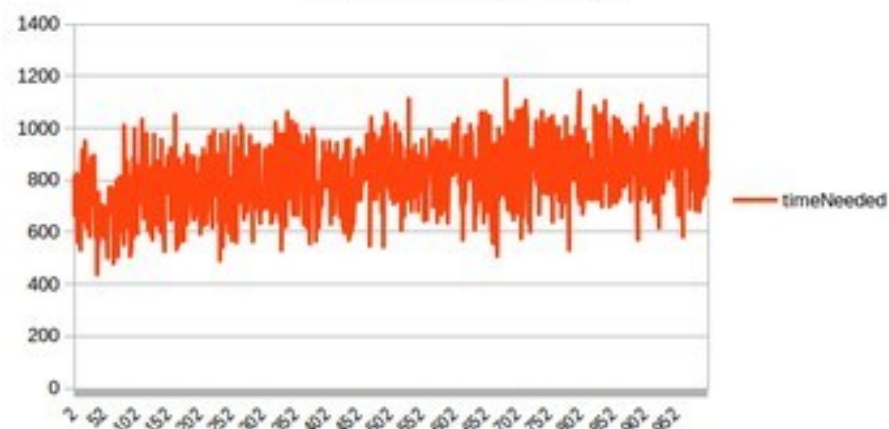
Predator

- Spawn 8 predators one at a time
- 10 % faster than prey, fix values for parameters
- Despawn after certain time or eating a prey
- Randomly selected surviving prey mutates into new generation

high mutate, weak pred, time



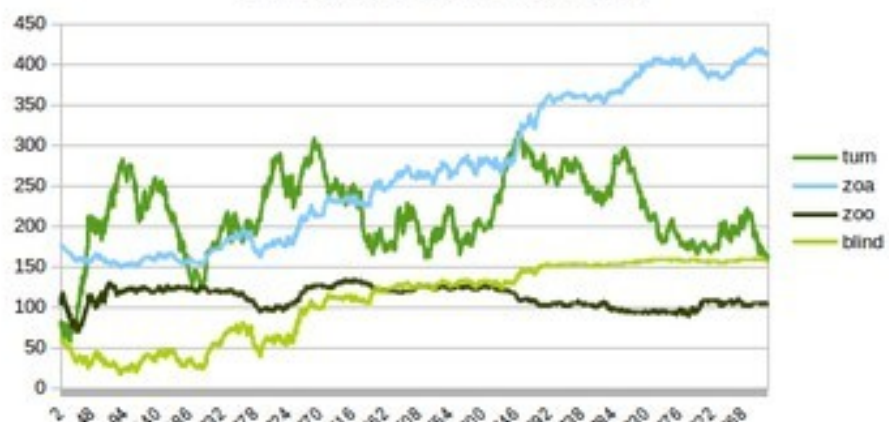
low mutate, weak pred, time



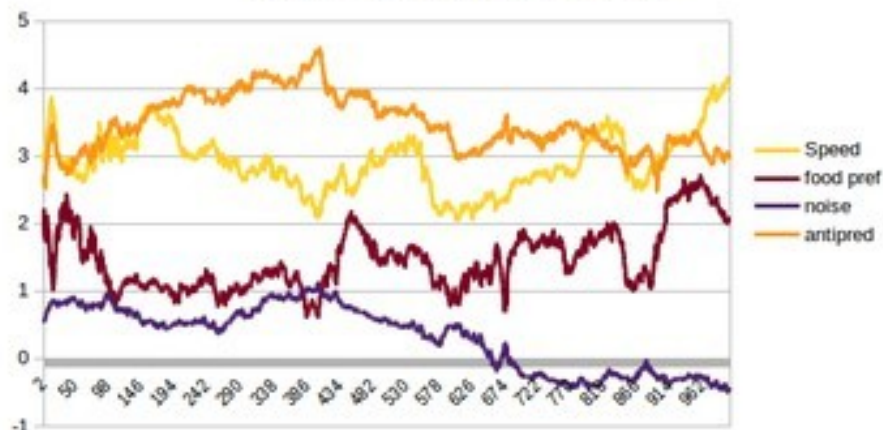
high mutate, weak pred, big params



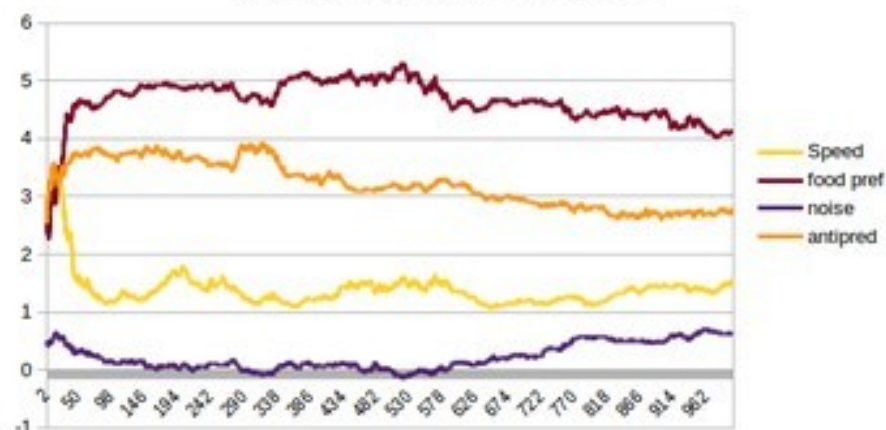
low mutate, weak pred, big params



high mutate, weak pred, smal params



low mutate, weak pred, small params



```
#-----new generation for predator simulation
def pickParent_simple_random(agents):
    r = random.randrange(len(agents))
    return agents[r]

# picks parents, creates new agents with mutated values,
# returns them in a new agents list
def nextGen_pred(agents):
    tempAgents = []

    for i in range(agentNum):
        parent = pickParent_simple_random(agents)
        child = mutate(parent)
        tempAgents.append(child)

    return tempAgents
```


Quantification

- Test cases

- Mutation values:

- High
 - Low (half of High)

- Parameters for predator

- Good: 140 turn angle, 45 blind angle
 - Bad: 70 turn angle, 135 blind angle

#mutate values

m_zoo_r = 5

m_zoa_r = 10

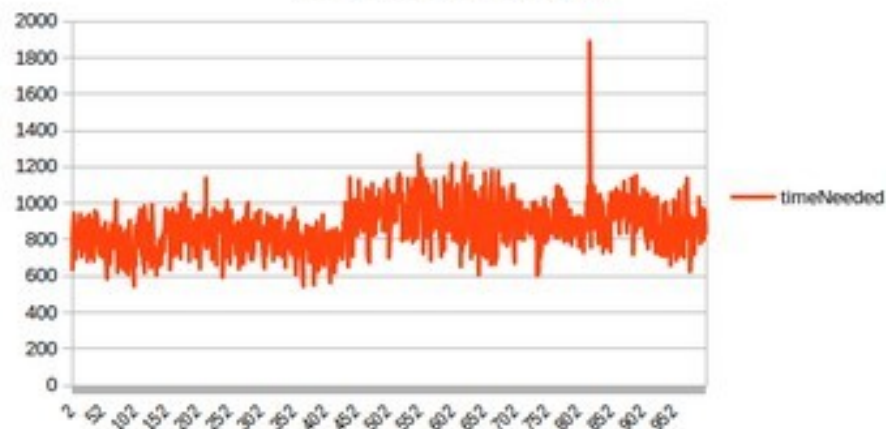
m_speed = 0.1

m_food = 0.1

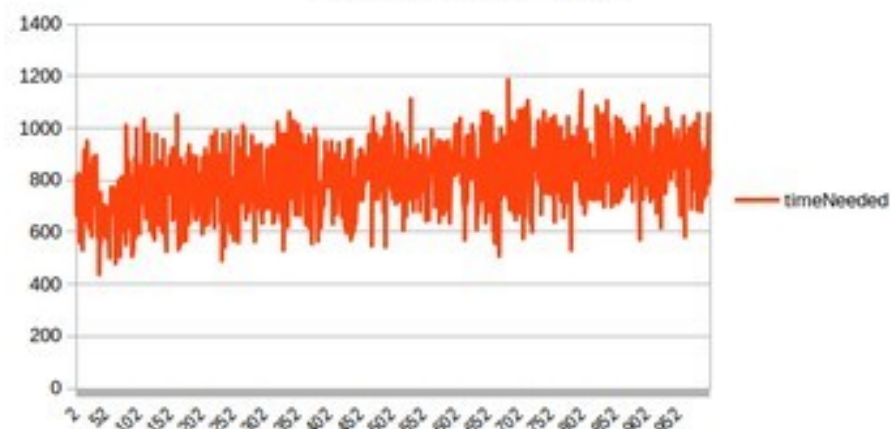
m_pred = 0.1

m_noise = 0.05

high mutate, weak pred, time



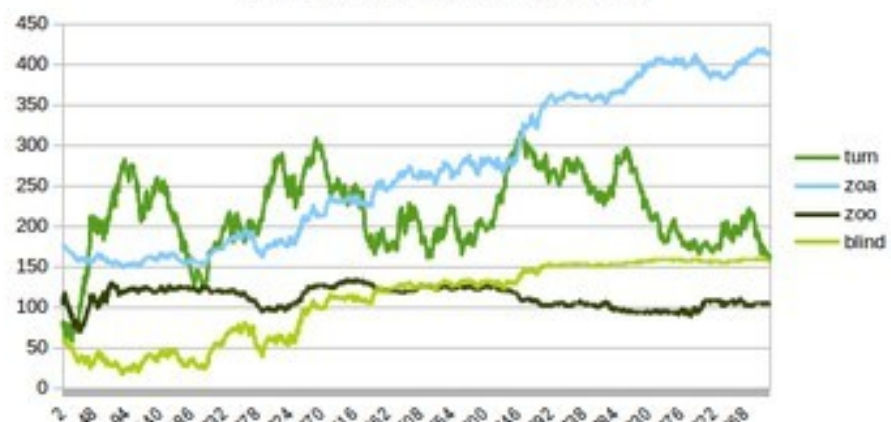
low mutate, weak pred, time



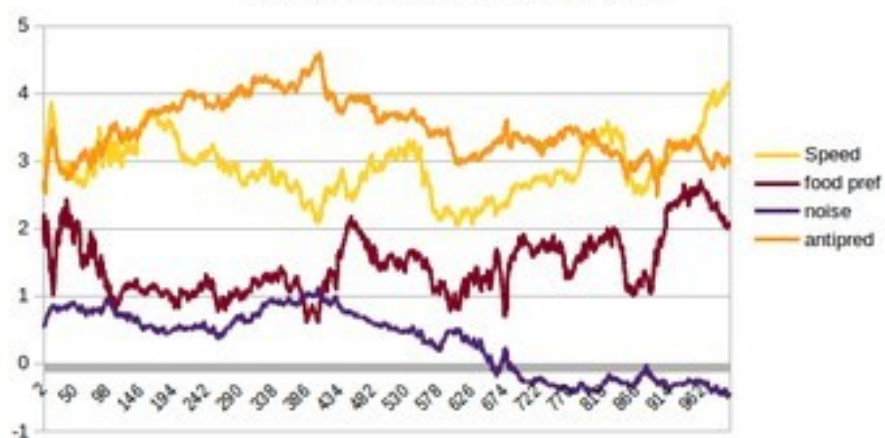
high mutate, weak pred, big params



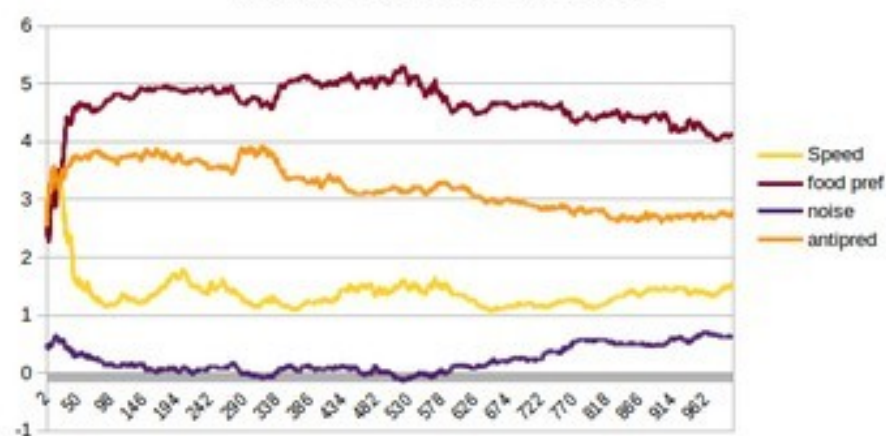
low mutate, weak pred, big params



high mutate, weak pred, smal params



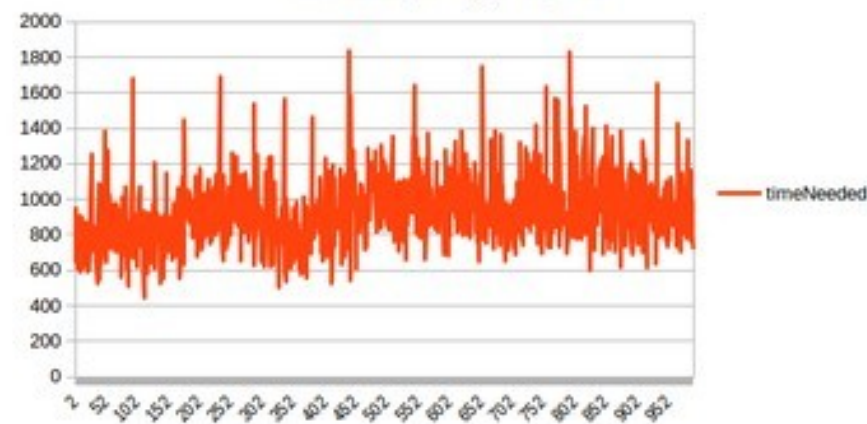
low mutate, weak pred, small params



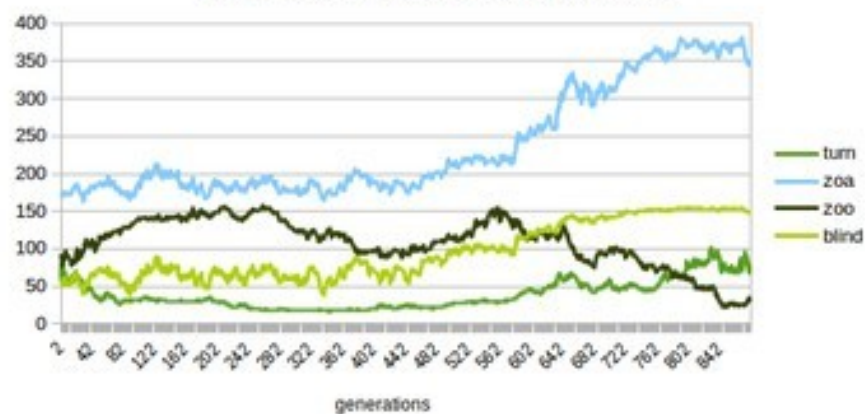
high mutate, strong predator, time



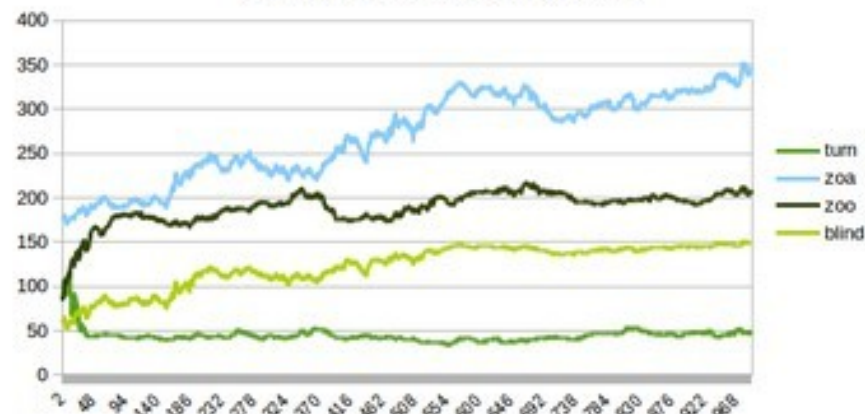
low mutate, strong pred, time



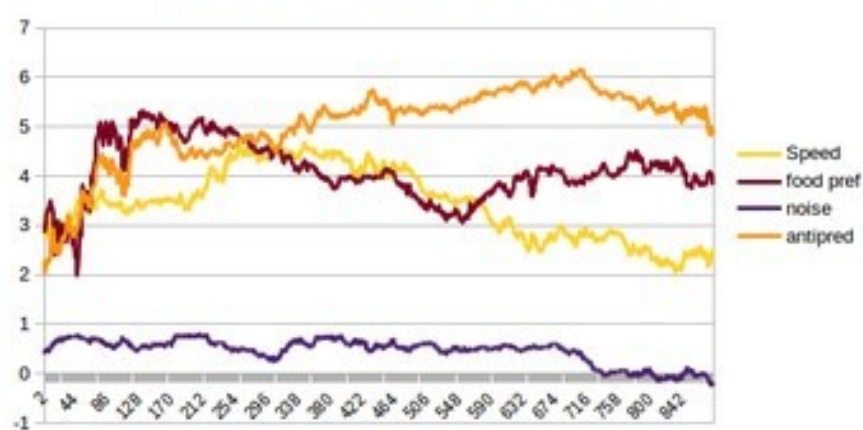
high mutate, strong predator, big parameters



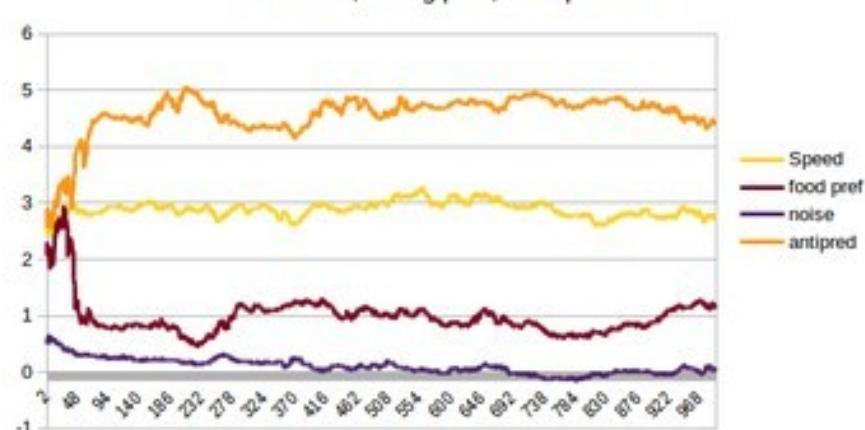
low mutate, strong pred, big params



high mutate, strong predator, small parameters



low mutate, strong pred, smal params



Food

- Food Particle spawn in cluster
- Borders limited in warm up
- Food spawns diagonal to the warm up area
- Iterations ends when $7/8$ of food eaten
- New generation, parent probability by amount of food eaten


```
# picks parents, creates new agents with mutated values,  
# returns them in a new agents list  
# for choosing parent
```

```
def generateWheel(agents):  
    wheel = []  
    currentsum = 0  
    for agent in agents:  
        currentsum += agent.foodlevel  
        wheel.append(currentsum)  
    return wheel
```

```
# chooses parent to mutate
```

```
def pickParent(agents, wheel):  
    r = random.randrange(wheel[-1]) + 1  
    for i in range(len(wheel)):  
        if r <= wheel[i]:  
            return agents[i]
```

```
def nextGen_food(agents):  
    wheel = generateWheel(agents)  
    tempAgents = []  
    for i in range(agentNum):  
        if wheel[-1] == 0:  
            parent = pickParent_simple_random(agents)  
        else:  
            parent = pickParent(agents, wheel)  
        child = mutate(parent)  
        tempAgents.append(child)  
    return tempAgents
```

Quantification

- Test cases:
 - Mutation values
 - High
 - Low(half of high)
 - Food clustering (128 total particles)
 - Big clusters (32)
 - Small(2)

#mutate values

m_zoo_r = 5

m_zoa_r = 10

m_speed = 0.1

m_food = 0.1

m_pred = 0.1

m_noise = 0.05

Food

