Preditor vs Prey (Pursuit) (COSC 4P82)

Kelvin Odinamadu*, Gideon Oludeyi†
Computer Science
Brock University
St. Catharines ON, Canada
*ko20so@brocku.ca †go21zq@brocku.ca

***Index Terms*—Genetic Programming, Predator-Prey Simulation, Artificial Intelligence (AI), Evolutionary Algorithms, Behavioral Strategy Evolution, Agent-based Modeling, Data Science**

I. Introduction

Our experiment aims to develop a genetic programming (GP) system that simulates a predator-prey scenario. The experiment is based on the artificial ant problem, to evolve strategies for effectively capturing moving prey within a simulated environment. We will explore and analyze how different GP language designs, changes in simulation conditions, and fitness evaluation criteria affect the efficiency of a predator in hunting its prey. The objective is to gain an understanding of predator-prey interactions, the development of complex strategies from simple rule sets, and the potential for genetic programming to model adaptive, intelligent behavior in a dynamically changing environment.

II. Experimental Setup

The GP system of the DEAP [1] Python package was used for the experiment. The experiment consisted of 10 runs with each run initialized with a different seed for the random number generator, and the results were averaged up.

A. Dataset

The experimental setup consists of a grid-based environment where predator and prey agents follow specific movement and interaction rules.

TABLE I
Simulation Parameters Definition

Parameter	Description	Type
Grid Size	Dimensions of the simulation grid	`int`
Predator Start Positions	Initial positions of all predators	`[(int, int)]`
Prey Count	Total number of prey in the simulation	`int`
Prey Start Positions	Initial positions of all prey	`[(int, int)]`
Simulation Steps	Total steps in each simulation run	`int`
Movement Rules	Defines how agents move within the grid	`String`

The simulation starts with the predator and prey placed at specific locations within the grid. The prey agents move randomly across the grid, while the predator's movement strategy evolves through genetic programming to maximize prey captures. The effectiveness of evolved predator strategies is evaluated over multiple simulation runs, each starting with a different seed for the random number generator to ensure diverse evolutionary outcomes.

B. Language

The GP language for the predator-prey simulation is designed to allow for complex predator behaviors by combining basic movement commands and decision-making based on sensory inputs. This set of functions lets the predator change dynamically to its environment and the movement of prey. Table II shows the core functions of the GP language used in the simulation.

TABLE II
GP Language for Predator

Function	Definition	Arity
Move Forward	Predator moves one step forward	0
Turn Left	Predator turns left	0
Turn Right	Predator turns right	0
IfPreyAhead	Executes action if prey is ahead	2
IfPreyBehind	Executes action if prey is behind	2
IfPreyLeft	Executes action if prey is to the left	2
IfPreyRight	Executes action if prey is to the right	2
Progn2	Sequentially executes two actions	2
Progn3	Sequentially executes three actions	3

C. Fitness Function

The fitness function in the predator-prey simulation is meant to assess how effective a predator's hunting strategy is. In each simulation run, a predator's fitness is based on the number of prey it captures. This measure promotes the development of efficient and adaptable hunting behaviors.

The fitness of an individual predator, denoted as i, in a simulation run R is defined as $f_R(i) =$ number of prey captured by predator i. This fitness value indicates the predator's performance with respect to its main objective in the environment.

To ensure diversity in the predator strategies and to avoid overfitting to specific prey behaviors or simulation setups, multiple runs with different initial conditions and prey movements may be conducted, and the fitness scores can then be averaged or aggregated to determine overall performance.

a) Algorithm Description: The algorithm "Algorithm 1" calculates the performance of predator agents within the simulation. It does this by averaging the number of successful

Algorithm 1 Predator Fitness Evaluation Algorithm

$totalPreyCaptured \leftarrow 0$
$simulationRuns \leftarrow$ Total number of runs
for $run = 1$ to $simulationRuns$ **do**
  Initialize environment with predators and prey
  $preyCapturedInRun \leftarrow 0$
  for each step in simulation **do**
    Execute actions for both predators and prey
    Update positions and check for captures
    if a predator captures prey **then**
      $preyCapturedInRun \leftarrow preyCapturedInRun + 1$
    end if
    Optionally update simulation state
  end for
  $totalPreyCaptured \leftarrow totalPreyCaptured + preyCapturedInRun$
end for
$fitness \leftarrow \frac{totalPreyCaptured}{simulationRuns}$
return $fitness$

prey captures across multiple simulation runs. The effectiveness of the predators in capturing prey directly influences their fitness, which indicates how well they hunt. Overall, the process involves selecting strategies that maximize prey captures, while working on natural selection within ecological systems.

TABLE III
ADAPTED GP PARAMETERS FOR PREDATOR-PREY SIMULATION

Parameter	Value (Adapted from Ant Simulation)
Population Size	100
Max. Tree Depth	7
Initial Tree Depth (min-max)	2-4
Tournament Size	5
Max. Generations	50
Crossover Probability	0.9
Mutation Probability	0.1
Elitism Rate	1
Selection Method	Tournament Selection

III. RESULTS

A. Performance run and comparison

This section analyzes how different environmental configurations, such as "wrap-around arenas" versus "arenas surrounded by impassable walls," impact predator behavior in a simulation.

1) Environmental Conditions:

a) Wrap-Around Arenas: In wrap-around environments, agents can seamlessly move from one edge of the simulation space to the opposite side, allowing for continuous movement and pursuit. Figure 1s setup encourages strategies that exploit the vast arena for effective prey tracking and capture. The program that demonstrates this strategy is as follows:

```
Progn2(
    IfPreyAhead(
        MoveForward,
        Progn2(
            IfPreyLeft(TurnLeft, TurnRight),
            MoveForward
        )
    ),
    IfPreyRight(
        TurnRight,
        IfPreyLeft(TurnLeft, MoveForward)
    )
)
```

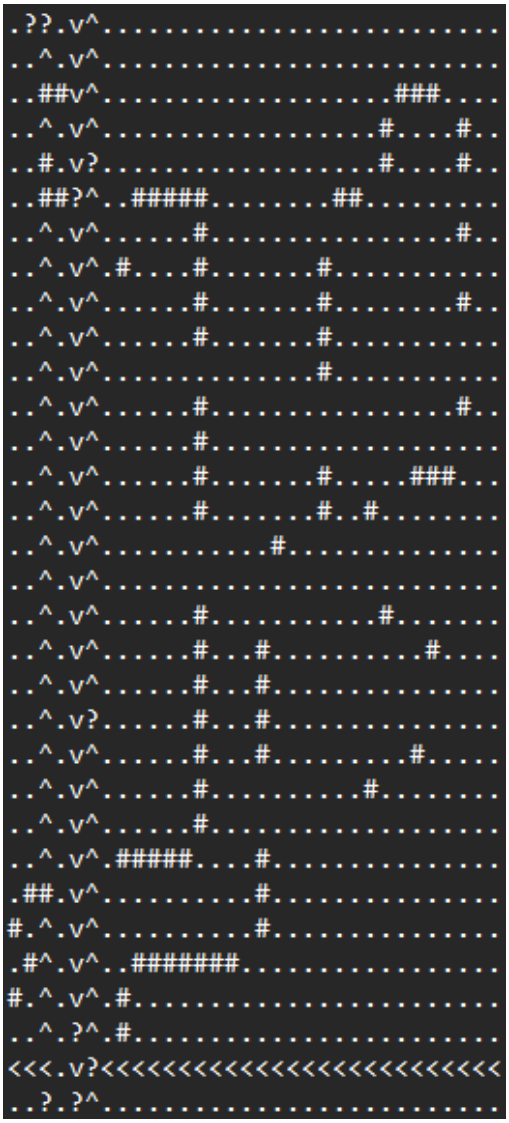

Fig. 1. Visual trace file for aggressive pursuit strategy without walled arena

b) Arenas with Surrounding Walls: Environments with walls make movement challenging as it is limited by impassable boundaries. Figure 2 constraints require strategies that consider spatial limitations, potentially using walls for ambush tactics or to prevent prey escape. The program is as follows:

```
Progn2(
    IfWallAhead(
        Progn2(
            IfPreyRight(TurnRight, TurnLeft),
            IfPreyLeft(TurnLeft, TurnRight)
        ),
        MoveForward
    ),
    Progn2(
        IfPreyAhead(MoveForward, TurnRight),
        IfNoPreyAhead(TurnLeft, MoveForward)
    )
)
```

TABLE IV
TRACE FILE SIMULATION REPRESENTATION

Description	Simulation character
Grid space	.
Predator	>
Prey	#
Predator catches prey	>?

Table IV, is the detailed description of the trace file simulation. For direction, ">" points the predator to the right, "<" to the left, "^" represents up and "v" represents downward direction.

2) Solution Analysis: The strategies evolved for each environmental setup showcase the diversity and adaptability of predator behaviors to distinct simulation challenges.

a) Solution 1: Aggressive Pursuit in Wrap-Around Arenas: In wrap-around arenas, as shown by Figure 1, predators developed aggressive pursuit tactics, which show high efficiency in capturing prey through relentless chasing, aided by the arena's boundless nature. Figure 3 shows an upward trend of the best and average fitness, which means over generations, the predator gets better at catching prey for this strategy in the wrapped arena.

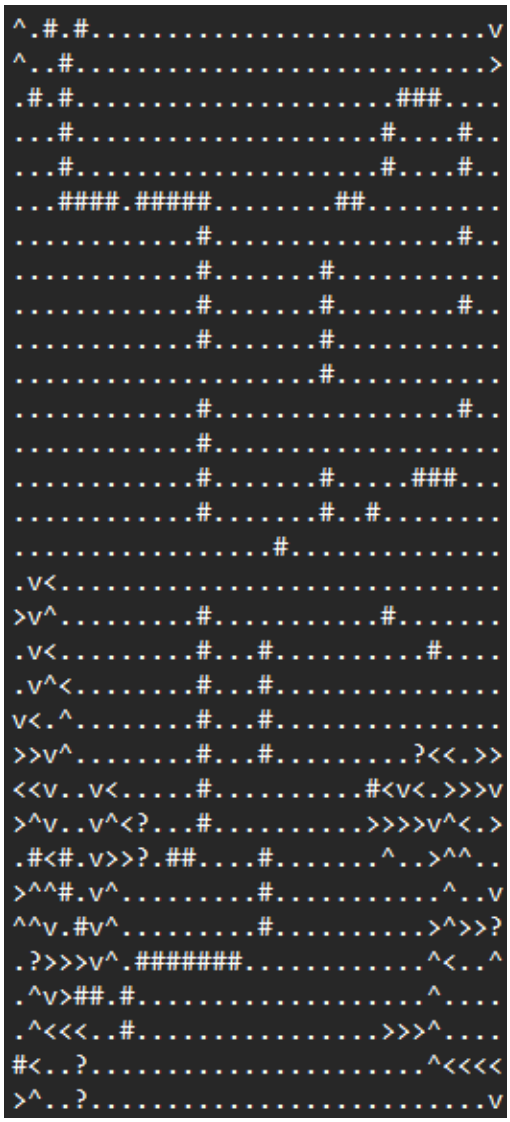

Fig. 2. Visual trace file for strategic ambush strategy with walled arena

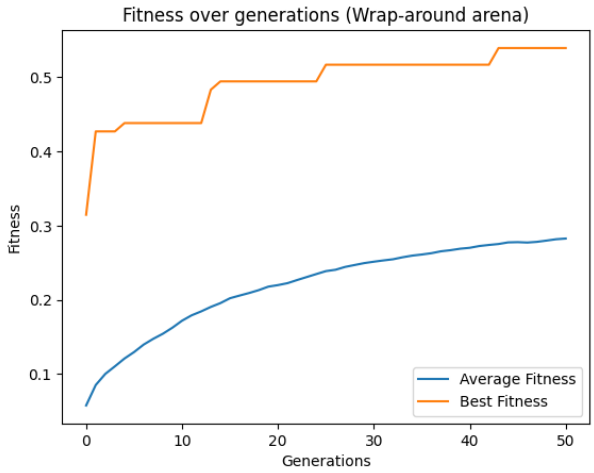

Fig. 3. Aggressive pursuit strategy in a wrap-around arena, emphasizing continuous movement across boundaries.

b) Solution 2: Strategic Ambush in Arenas with Walls: In contrast, predators in walled arenas adapted to use strategic ambush tactics, as shown by Figure 2, leveraging the environmental barriers to enhance their hunting success by cornering or intercepting prey. Figure 4 also shows an upward trend in the best and average fitness, meaning the predator gets better at catching prey over generations, for this strategy, in the walled arena simulation.

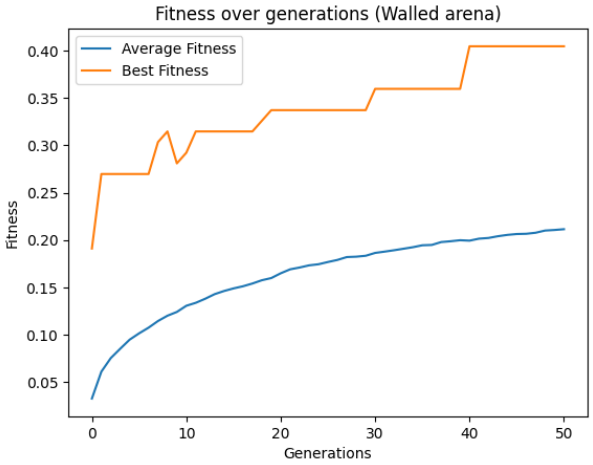

Fig. 4. Strategic ambush strategy in an arena with walls, utilizing environmental constraints for effective predation.

3) Discussion: The table below (Table V) shows how well the predator performed in different simulation runs.

a) Walled Arena Performance: In walled arenas, predators encounter some challenges: - **Average Capture Rate (16.34%):** This lower average capture rate suggests that predators in a walled arena, find it difficult to track and capture prey. This is because the impassible boundaries limit the predator's ability to pursue prey directly, which requires more complex navigation and strategic adaption.

- **Best Capture Rate (40.44%):** The best performance in walled arenas shows that though the environment has challenges, effective strategies can still show up. These strategies might look like using the walls to corner prey or using the environment's geometry to predict and intercept prey movements.

b) Wrap-around Arena Performance: On the other hand, wrap-around arenas offer a more open environment for predators: - **Average Capture Rate (21.92%):** The higher average capture rate in wrap-around arenas suggests that predators benefit from the boundless space, making pursuit more straightforward and likely reducing the complexity of tracking moving prey. - **Best Capture Rate (53.93%):** The significant increase in the best capture rate compared to walled arenas highlights the advantage of a wrap-around environment in optimizing predation strategies. The boundless arena likely encourages more aggressive and direct pursuit tactics, allowing higher efficiency in catching prey.

TABLE V
SUMMARY OF RUNS

	Walled Arena	Wrap-around Arena
Average	16.34%	21.92%
Best	40.44%	53.93%

IV. CONCLUSION

Our research focused on how different simulated environments affect predator-prey strategies, using genetic programming. We found that environmental settings play a key role in shaping evolutionary outcomes and illustrated the adaptability of GP in modeling complex interactions within these systems.

The research showed that predators' efficiency in capturing prey varied significantly between walled and wrap-around arenas, revealing the critical role of environmental constraints in evolutionary processes. Specifically, predators in wrap-around arenas exhibited higher capture rates, suggesting that less restricted movement aids in more effective pursuit behaviors.

To further explore this topic, we need to consider the environmental constraints, how agents interact with each other, and the potential for predators and prey to develop strategies together. In the future, we can improve the model by including different environmental factors, using advanced computational methods to refine strategy evolution, and creating interactive visualizations to make the simulation more engaging, similar to a Pac-Man game.

This research sets the foundation for further study in computational ecology. It focuses on how different environments impact the survival strategies of predators and prey. Building on this could provide deeper insights into the inner workings of ecological systems and the potential of genetic programming to explore them.

V. APPENDIX

Best solution across runs:

```
ifFoodAhead(ifFoodAhead(prog3(ifFoodAhead(forward
forward), ifFoodAhead(ifFoodAhead(ifFoodAhead(ifFo
right), prog3(right, right,
left)), prog3(ifFoodAhead(left,
right), prog3(left, right, left),
ifFoodAhead(forward, right))),
prog2(prog3(prog2(right, right),
prog3(forward, left, right),
prog3(forward, forward, right)),
prog2(prog2(forward, right),
prog3(right, right, left)))),
prog2(ifFoodAhead(ifFoodAhead(prog2(left,
forward), prog2(forward, right)),
prog3(ifFoodAhead(forward, right),
prog3(left, right, right),
prog3(right, forward, right))),
ifFoodAhead(ifFoodAhead(ifFoodAhead(forward,
right), prog2(forward, right)),
ifFoodAhead(prog3(left, left, right),
ifFoodAhead(right, left)))))),
prog2(ifFoodAhead(prog3(prog2(prog3(forward,
right, left), prog3(forward, left,
forward)), ifFoodAhead(prog2(right,
forward), ifFoodAhead(right,
forward)), prog2(ifFoodAhead(left,
left), prog2(forward, right))),
prog3(prog2(prog2(forward, right),
prog3(forward, right, forward)),
ifFoodAhead(prog3(left, right,
right), ifFoodAhead(right,
right)), ifFoodAhead(prog2(left,
```

```
forward), prog2(right, right)))),
prog2(prog2(prog3(ifFoodAhead(right,
right), prog2(left, forward),
prog3(forward, forward, right)),
prog3(prog3(forward, left, right),
ifFoodAhead(right, left), prog2(left,
right))), prog2(prog3(ifFoodAhead(left,
left), prog2(forward, left),
prog3(left, forward, forward)),
ifFoodAhead(prog2(forward, left),
prog2(right, right))))), left),
prog3(ifFoodAhead(forward, forward),
prog2(left, right), forward)), left),
ifFoodAhead(prog3(prog2(right, left),
ifFoodAhead(forward, right), left),
ifFoodAhead(ifFoodAhead(forward,
prog2(prog2(prog3(prog3(prog2(ifFoodAhead(left,
forward), ifFoodAhead(left,
right)), right, forward),
ifFoodAhead(prog3(prog3(forward,
left, left), prog2(forward, forward),
ifFoodAhead(ifFoodAhead(ifFoodAhead(prog3(ifFoodAhead(prog3(prog2(right),
right), prog3(left, left, right)),
ifFoodAhead(right, ifFoodAhead(forward,
forward)), ifFoodAhead(prog2(right,
right), ifFoodAhead(left, forward))),
prog3(prog2(prog3(right, left, forward),
ifFoodAhead(ifFoodAhead(prog3(left,
prog3(forward, right, forward), left),
prog3(right, left, prog2(left, left))),
right)), prog3(ifFoodAhead(forward,
right), prog2(forward, prog2(right,
right)), prog3(right, left,
right)), prog2(prog3(forward, left,
ifFoodAhead(prog3(left, left,
right), ifFoodAhead(right, right))),
prog3(forward, right, right)))),
prog3(ifFoodAhead(prog3(prog2(right,
forward), prog3(forward, right,
left), ifFoodAhead(left,
ifFoodAhead(ifFoodAhead(prog3(ifFoodAhead(right,
forward), prog3(right,
prog3(ifFoodAhead(left, forward),
ifFoodAhead(left, left), ifFoodAhead(left,
right)), forward), prog3(left, forward,
right)), prog2(prog3(left, forward,
right), ifFoodAhead(right, right))),
prog2(ifFoodAhead(prog3(forward,
left, left), prog2(right, left)),
ifFoodAhead(prog2(right, left),
prog2(right, forward))))))),
prog3(ifFoodAhead(left,
right), prog2(right, forward),
prog2(left, right))), forward,
prog2(ifFoodAhead(ifFoodAhead(forward,
right), prog2(forward, left)),
prog2(ifFoodAhead(right, forward),
prog2(forward, right)))),
prog3(ifFoodAhead(prog3(ifFoodAhead(left,
forward), ifFoodAhead(left,
left), ifFoodAhead(left, right)),
prog2(ifFoodAhead(right,
right), prog2(right, right))),
ifFoodAhead(prog2(prog3(forward,
left, forward), prog3(left, right,
right)), ifFoodAhead(forward, left)),
prog2(prog3(ifFoodAhead(ifFoodAhead(prog2(prog3(ri
left, forward), ifFoodAhead(prog2(right,
right), ifFoodAhead(ifFoodAhead(forward,
forward), forward))), prog2(left,
forward)), forward), ifFoodAhead(forward,
left), prog2(forward, left)),
prog2(prog3(prog3(left,
right, right), left, left),
prog3(right, forward, left))))),
prog2(ifFoodAhead(ifFoodAhead(prog3(prog2(forward,
forward), prog3(left, left, right),
ifFoodAhead(prog3(prog2(right),
prog2(prog3(forward, left, right),
ifFoodAhead(right, left))),
ifFoodAhead(prog2(prog2(left,
forward), prog3(forward, right,
right)), prog3(ifFoodAhead(right,
left), ifFoodAhead(forward,
forward), prog2(right, right)))),
ifFoodAhead(prog3(prog3(prog3(left,
forward, right), prog3(left,
right, right), prog2(forward,
prog3(forward, forward, right))),
ifFoodAhead(ifFoodAhead(left, left),
prog2(left, left)), prog3(prog3(left,
left, forward), prog3(right, left,
left), prog2(forward, forward))),
prog3(prog3(ifFoodAhead(left, left),
prog2(forward, forward), prog3(left,
left, right)), prog2(ifFoodAhead(forward,
forward), ifFoodAhead(right, left)),
ifFoodAhead(ifFoodAhead(right, left),
ifFoodAhead(right, right))))),
ifFoodAhead(prog3(ifFoodAhead(prog3(prog2(left,
left), prog2(prog2(right,
forward), prog2(left, right)),
ifFoodAhead(prog3(forward, left,
left), prog2(right, right))),
prog3(prog2(prog3(right, forward,
forward), prog3(right, right, left)),
prog3(prog2(left, left), prog2(right,
right), prog2(right, right)),
ifFoodAhead(ifFoodAhead(right, left),
prog3(left, forward, forward)))),
prog2(ifFoodAhead(ifFoodAhead(prog3(right,
forward, right), prog2(left, right)),
```

```
ifFoodAhead(ifFoodAhead(forward,
right), ifFoodAhead(left, left))),
prog2(prog3(prog2(left, right),
prog2(left, left), forward),
prog2(prog3(right, right,
forward), prog2(forward, left)))),
ifFoodAhead(ifFoodAhead(prog3(prog2(right,
forward), prog3(forward, forward,
left), prog2(left, forward)),
ifFoodAhead(prog3(right, right,
right), ifFoodAhead(left, right))),
prog2(prog3(ifFoodAhead(forward,
prog2(forward, right)), prog3(left,
forward, right), prog2(left, right)),
ifFoodAhead(ifFoodAhead(forward,
forward), prog2(forward, right))))),
prog2(prog3(prog2(prog2(prog3(forward,
left, forward), ifFoodAhead(forward,
left)), prog2(prog3(right, right,
left), prog3(forward, right, left))),
ifFoodAhead(prog3(ifFoodAhead(left,
forward), prog2(left, right), prog3(left,
forward, right)), prog2(prog3(right,
left, right), ifFoodAhead(right,
left))), ifFoodAhead(prog3(prog2(right,
right), prog2(forward,
forward), prog2(right, left)),
prog2(prog3(left, forward, forward),
ifFoodAhead(forward, forward)))),
ifFoodAhead(prog3(ifFoodAhead(ifFoodAhead(right,
forward), ifFoodAhead(left, left)),
prog3(prog3(forward, forward, right),
prog2(right, right), ifFoodAhead(left,
forward)), ifFoodAhead(ifFoodAhead(right,
right), prog2(prog2(prog2(prog2(forward,
right), left), ifFoodAhead(right,
prog2(right, left))), forward))),
prog3(prog3(prog2(left, forward),
prog2(left, right), prog2(forward,
left)), prog3(prog3(left, forward, right),
prog2(right, right), prog3(left, right,
left)), prog3(prog3(forward, forward,
prog2(left, right)), prog2(forward,
left), prog2(left, right)))))))),
right)), prog3(prog2(forward,
left), left, prog3(forward,
forward, right))), prog2(right,
right)), ifFoodAhead(prog2(forward,
left), prog2(left, left))),
ifFoodAhead(ifFoodAhead(prog3(forward,
forward, prog2(forward, right)),
ifFoodAhead(left, prog2(prog3(prog2(right,
right), prog3(forward, left,
right), prog2(forward, forward)),
prog3(ifFoodAhead(forward, right),
prog3(right, left, right),
ifFoodAhead(right, left))))),
prog3(right, ifFoodAhead(right, forward),
ifFoodAhead(forward, forward))))),
ifFoodAhead(right, forward))))
```

REFERENCES

[1] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," Journal of Machine Learning Research, vol. 13, pp. 2171–2175, jul 2012.