Automatic and Genetic Programming

Discussion lead by Julieta
Some remarks, and goals that don’t fit perfectly in each reading

INTRODUCTION
Reviewing some learning goals

• What is the difference between **Automatic** and **Genetic Programming**?
  – Automatic Programming is **the task** of creating programs automatically
  – Genetic Programming is **a method** to create programs automatically

• What is the difference between Genetic **Algorithms** and Genetic **Programming**?
  – Genetic Programming evolves **algorithms**
  – Genetic Algorithms evolve **instance** solutions
  – This can get very, very **blurry**
Or why cooperation is good.

A STRATEGY OF WIN-STAY, LOSE-SHIFT THAT OUTPERFORMS TIT-FOR-TAT IN THE PRISONER'S DILEMMA GAME.
Evolutionary Game Theory

• Outline
  – Where did it come from?
  – Evolutionary Stable Strategies (ESS)
  – The Prisoner’s dilemma in Evolutionary Game Theory
  – Why this paper is a big deal
  – Puzzle! Fitness evaluation
Where did it come from?

1940s

John von Neumann

Oskar Mortgensen

1950s

John Nash

Reinhard Selten

1970s

John Maynard Smith

George Price
Evolutionary Stable Strategies

• Evolutionary Game Theory is like Game Theory, except that your opponents can evolve!

• The main question is
  – Can an alternative invade?
  – If not, then it is an Evolutionarily Stable Strategy (ESS)

Just like in Pokemon!

This is equivalent to a Nash Equilibrium
### The Prisoner’s dilemma

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C</strong></td>
<td>{R, R}</td>
<td>{S, T}</td>
</tr>
<tr>
<td><strong>D</strong></td>
<td>{T, S}</td>
<td>{P, P}</td>
</tr>
</tbody>
</table>

- **T > R > P > S**
- **2R > T + S**

- **Betraying pays off**
- **Sucker**
- **Globally, it’s better to cooperate**
Why is this paper a big deal?

• This problem has tons of applications in sociology, economics, psychology...

• The **Iterated** Prisoner’s Dilemma
  – You play the prisoner’s dilemma *over and over* again
  – In the short term, you want to *defect*, but overall you want to *cooperate*
  – **Axelrod**’s Tournament
  – **TFT** won!
1940s

Robert Axelrod

1950s

Martin Nowak

1970s

Karl Sigmund

1980s

1990s and on…
Puzzle! How do we evaluate fitness?

• As someone pointed out on Piazza, this simulations can be very, very expensive
• How would you evaluate fitness?
• Let’s get carried away with the Pokemon analogy!
Wild Tit for Tat appeared!
Tit For Tat

Pavlov, go!

Pavlov
Tit for tat used COOPERATE!

Pavlov, used DEFECT!
Tit For Tat

And so on...

Pavlov won 5 fitness points!

Tit for Tat Blacked Out!

Pavlov
Puzzle! How do we evaluate fitness?

• This would take forever!
• The paper mentions another way to simulate iterated prisoner’s dilemmas...
• Each player is defined by a vector \((R, S, T, P)\) of probabilities to cooperate according to the payoff of the last iteration
• This can be simulated efficiently using a Markov chain!
• What does the transition matrix look like?
<table>
<thead>
<tr>
<th></th>
<th>CC</th>
<th>CD</th>
<th>DC</th>
<th>DD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CC</strong></td>
<td>(r_1 \times r_2)</td>
<td>(r_1 \times (1 - r_2))</td>
<td>((1 - r_1) \times r_2)</td>
<td>((1 - r_1) \times (1 - r_2))</td>
</tr>
<tr>
<td><strong>CD</strong></td>
<td>(s_1 \times t_2)</td>
<td>(s_1 \times (1 - t_2))</td>
<td>((1 - s_1) \times t_2)</td>
<td>((1 - s_1) \times (1 - t_2))</td>
</tr>
<tr>
<td><strong>DC</strong></td>
<td>(t_1 \times s_2)</td>
<td>(t_1 \times (1 - s_2))</td>
<td>((1 - t_1) \times s_2)</td>
<td>((1 - t_1) \times (1 - s_2))</td>
</tr>
<tr>
<td><strong>DD</strong></td>
<td>(p_1 \times p_2)</td>
<td>(p_1 \times (1 - p_2))</td>
<td>((1 - p_1) \times p_2)</td>
<td>((1 - p_1) \times (1 - p_2))</td>
</tr>
</tbody>
</table>
Remarks

• Now that you know all this...

• I suggest you go back and re-read the 3-page paper. You’ll enjoy it much more!
Remarks

- Feel free to play with my implementation, available at
AUTOMATED DISCOVERY OF LOCAL SEARCH HEURISTICS FOR SATISFIABILITY TESTING
• Outline
  – A historical perspective on local-search algorithms
  – Remarks found by Fukunaga that motivate an automated process for heuristic discovery
  – How is the performance evaluated?
  – What are some disadvantages of this approach?
• **All** local-search SAT heuristics look like this

1. $A := \text{randomly generated truth assignment}$
2. For $j := 1$ to cutoff
3. If $A$ satisfies formula then return $A$
4. $V := \text{Choose a variable using a variable selection heuristic}$
5. $A := A$ with value of $V$ flipped
6. Return FAILURE (no satisfying assignment found)

• Some **handy** definitions
  – (net, positive, negative) gain
  – Variable age
A historical perspective on local-search SAT heuristics

- 1992: GSAT
- 1993: HSAT
- 1993: GWSAT
- 1994: Walksat
- 1997: Novelty
- 1999: Novelty+
- 1997: R-Novelty

This one starts to look quite complicated to me
• They are all variations on a theme
  – Scoring of variables
  – Variable selection
  – Ranking of variables and greediness
  – Variable age
  – Branching

• “The history of SAT local search algorithms shows that significant advances do not require the invention of entirely new “ideas” – discovering a new combination of existing building blocks has resulted in some of the best known SAT local search algorithms”.
• “The task of combining building blocks into successful variable selection heuristics is difficult to perform manually, even for expert researchers, and is well-suited for an automated system”.

• CLASS
  – An s-expressions language
  – A meta-algorithm that combines programs
What does **Evaluate** do?

“A candidate heuristic is executed on a set of training instances from the distribution of hard, randomly generated 3-SAT problems from the SAT/UNSAT phase transition region”

\[(\# \text{ of 50-var successes}) + 5 \times (\# \text{ of 100-var successes}) + \frac{1}{\text{MeanFlipsInSuccessful Runs}}\]
You run the algorithm for a while and you get things like this!

```
(OLDER-VAR
 (OLDER-VAR
  (IF-TABU 5 (GET-VAR +BC0+ +NET-GAIN+)
  (GET-VAR +BC1+ +RANDOM+))
 (IF-VAR-COMPARE <= +NEG-GAIN+
   (RANDOM-VAR +BC0+)
   (RANDOM-VAR +BC1+)))
 (IF-VAR-COMPARE <= +NET-GAIN+
  (IF-TABU 5 (GET-VAR +BC1+ +POS-GAIN+)
  (GET-VAR +BC0+ +NEG-GAIN+)
  (IF-VAR-COMPARE <= +NEG-GAIN+
    (GET-VAR +BC1+ +NEG-GAIN+)
    (GET-VAR +BC0+ +NET-GAIN+)))))
```
Disadvantages?

• Solutions grow too quickly
• The solutions are rather unintuitive
• It’s quite time-consuming