# Genetic Object Oriented Programming

(GOOP)

# Outline

- Basic idea
- Previous work
- What might we get?
- Reality check
- Possible features

# Basic idea

- -Genetic Algorithms: Arguable good search strategy. But generally done on bit strings or the like which don't translate easily to solution space.
- -Genetic Programming: Applies LISP like syntax to GAs so they can solve more general problems with less translation work.
- -Object Oriented Programming: Useful abstraction for human programming.
- Can we combine Genetic Programming and Object Oriented Programming and get anything useful?

# Previous work

- Koza, John R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection.
- -Individuals trees of LISP operations
- -Each operation closed so we can apply any in any order
- -Mutation changes operations and operands-Crossover just copies on section of the tree to another individual
- -To evaluate just run expression
- Used with success in real world problems

# Previous work

- Montana, David (1995) Strongly Typed Genetic Programming.
- -Based on GP
- -But operations not closed to one type of value
- -So individuals need to be moderated so each operator returns correct value for parent
- -Mutation and crossover need to respect this property.
- Article claims STGP reduces search space and increases performance but is not without its drawbacks and needs further work.

# What might we get?

- Advantages of GP
- But add
- -Explicit modularity
- -Allow more insight into current solution at each level

# Reality check

- Can we do anything like OO?
- Sure we can just pretend the basic tree structure is an object hierarchy.
- Each object has a "execute" method which calls execute on the objects contained in it.
- This is kind of silly as it basically just renames GP but at least we know we have a trivial default which is equivalent to GP.
- We can also do the same thing for strongly typed GP just have each execute method require specific types for inputs.

# Possible features

- Explicitly named functions:Basic GP has no way of preserving some group of operations that are useful and reusing them in multiple locations (though crossover does this to some degree).
- Each object could be born with named methods which could mutate etc... and be used in the objects execute method. "parent objects" could then make use of the methods.
- Problem: what if crossover moves child out from under and object and it was using a method in that object.

# Possible features

- Try sequential programming
- Construct some objects (maybe as above) call objects methods on random objects storing results as you go.
- Pass these stored inputs into next calls randomly.
- Eventually return a call to a method that returns appropriate response.
- Mutations could add or remove function calls etc...
- Entire problem could be constructed this way if desired.

# Possible features

- Context evaluation: (stolen from Richard)
- Using the above sequential design we could produce root level in which we could evaluate objects.i.e. pick several sequences (perhaps highest fitness) and the switch in and out objects see what happens to fitness.

# Possible features

- Solution is object:We could try having the solution be an object with methods on it.
- For starters these would be our basic operators
- But we could try allowing the GOOP to find insightful methods on the object. Which can then be used.

# Possible features

- Whatever people can come up with :)