everything you wanted to know about computers*

John Alex

*but were too afraid to ask
Overview

• Functions as representation
• Function optimization methods, issues
  – Steepest Descent
  – Simulated Annealing
  – Genetic Algorithms
• Analysis of shape grammars
• Possibilities
Ode to Functions

• Math is based on them
• Computers are based on them
• Very general representation: a mapping
• Helpful as intermediate object too
  – aid to formalization, rigor

• Limited
  – only maps numbers to numbers
  – is mapping it?
Functions

• $y = f(x)$
• $x,y$: vector of parameters (‘parametric’?)

• “Form function”
  – Vertices = $f(\text{dimensions, key pts, etc})$

• “Fitness function”
  – Quality = $f(\text{vertices})$
Functions

• $y = f(x)$
• $x, y$ are each a vector of parameters

• Each parameter can be either
  – discrete (combinatorial): 0, 1, 2, 3, 4
  – continuous: 0-4
Functions

• $y = f(x)$
• such that $c(x) = 0$

• Constraints: valid parameter combinations
Trouble with Functions in Design

• Pre-Optimization questions:
  – how to define a useful form function?
    • Vertices = f(dimensions, key pts, etc)
  – how to define a useful fitness function?
    • fitness = f(geometry only)?

  – generality vs. specificity
  – myth: computer functions can be random
  – myth: designers’ functions are random
Trouble with Functions in Design

• Optimization question:
  – how to find the most fit form?

• Pre, mid, post-optimization question:
  – how to handle emergence?
    • changing form, fitness functions during design
    • changing question in middle of trying to answer it
Architectural Function Optimization

- Vertices = f(model parameters)
- Quality = f(vertices)
- Quality = f(model parameters)

- Optimization = vary model parameters to maximize goodness
Function Optimization

$$\max q(p_0, p_1, \ldots, p_n)$$
Keeping $c_0(p_0, p_1, \ldots, p_n) = 0$
and $c_1(p_0, p_1, \ldots, p_n) = 0$ and...

$$\max q(p)$$
keeping $c(p) = 0$
given some initial state $p_0$
Function Optimization

$$\text{max } q \ (p)$$
keeping $$c(p) = 0$$

$$p_0$$: initial state

- Goodness must be a single number
  - “multi-objective optimization”
- The problem: given start, where to go next?
  - in which direction?
  - how far?
Functions – Moving Around

- Move in a *direction* for a certain *distance*
- Start: (0,0)
- Move to (2,2): moving in (1,1) direction for a distance of 2.

- ‘Direction’ = a change in each parameter
- ‘Distance’ = multiplier of a direction
Functions – Continuity

• Changing continuous parameters: nicer
  – correlation: direction (1,1) roughly equals combination of effects from directions (1,0) and (0,1)
  – correlation: moving further along (.5, 4.2) tends to produce more of what that direction does
  – derivatives: can determine effect of parameter change without trying all possible changes

• Combinatorial: no correlation between directions, between distance and outcome
  – (usually)
  – shape grammars are combinatorial
Optimization Techniques

- Steepest Descent
- Simulated Annealing
- Genetic Algorithms
Hill-Climbing/Steepest Descent

• Go downhill
  – best downhill direction: downhill for each parameter
  – distance: select best along direction
  – can’t go downhill? Stop.

• Local or global extrema?
  – fundamental problem
Simulated Annealing

• Jump around…
• Jump around…
• Jump up, jump up….
• …and get down.
Simulated Annealing

• Jump around/up...
  – provisionally jump around a distance j (‘taking’ the new position if it’s better - or not much worse)

• …and get down
  – steadily lower jump distance j and uphill tolerance t
    • lower the energy of the system
    • steel cooling, water flooding
  – t = 0, small j: must get better at each step = hill-falling
Genetic Algorithms

• GA concepts can be thought of functionally:
  – Phenotype = f(genotype)
    • form = f(dimensions)
  – Fitness = f(phenotype)
    • fitness = f(form)

  – Genotype = dimensions
  – Phenotype = form
Genetic Algorithms

- Evolution as functional optimization
  - arbitrary directions
    - generated by:
      - interleaving parameters of parents (crossover)
      - ‘random’ change of parameter (mutation)
    - ignore direction, distance correlations
    - assume crossover groups are independent of each other
    - generates invalid parameter sets
    - locking down substrings: when? how many? for how long?
  - multiple kids: parallel optimization
    - i.e., multiple directions at once
  - unknown improvement in kids
    - have to evaluate fitness of each kid after generating it
  - possibly no improvement in kids (convergence?)
Shape Grammars - Generation

Generate all shapes \( s(\text{rules}, s_0, n) \)

- \( s_0 \): initial shape
- rules: shape rules
- \( n \): number of iterations

- Fitness function inside human operator
- Human optimizer changes parameters to increase fitness
- Implicit fitness/recognition function
Shape Grammars - Generation

- \((r_0, r_1, r_2, \ldots) = \text{recognizer}(\text{shape}, \text{set of all rules})\)
  - recognizer compares left sides, allowing for translation, scale, and rotation
  - recognizer as local fitness function: only certain rules are fit for this situation
- \((r_0, r_1, r_2, \ldots)\) are a set of (equally good!) directions
- Go in all directions, distance 1 (apply all valid rules once), producing \(\text{shape}_0, \text{shape}_1, \ldots, \text{shape}_n\)
- Recurse on all the kid shapes
Shape Grammars - Optimization

\[
\text{max } q \left( s_0, r_0, r_1, r_2, \ldots, r_n \right) \\
\text{keeping } c(s_0, r_0, r_1, r_2, \ldots, r_n) = 0 \\
\]

\( s_0 \): initial shape \\
\( n \): number of rules to apply

• Combinatorial representation 
  – regardless of fitness function, no good way to search the space
The Tough Questions

- Form function: What bogus forms are allowed? What useful forms are *not* allowed? What framework is embodied in the function?
- Fitness function: What does ‘fitness’ mean? (Does it encode architectural knowledge? How?)
- Representation of emergence?
- Optimization: Is it finding the global minimum (by starting near solution or being a bowl-shaped function)? If not, what is it finding?
- Recognition = binning based on sliding qualities
  - ‘x is awfully chair-like’ -> ‘x is a chair’
  - ‘x is somewhat chair-like’ -> ‘x is a chair’??
  - When is x not a chair? Who’s deciding, and how?
Computers in Design: Future

• support quick, narrow optimizations
  – support form changes
    • quick specification of formwork as it changes
  – brainstorming – show all combinations
• more fine-grained study of frameworks and how they change
  – given explicit representation, computer can help
Difficult Things Computers Do

• Simulation
  – light
  – structural strength
  – sound
  – heat

• Visualization
  – realistic: simulation of light
  – non-realistic: arbitrary artistic techniques to convey information
    • false color, overlays, collage, false perspective
    • show temperature, airflow, etc

• Calculation
  – area, cost, number of parts